UN ENVIRONNEMENT G-DEVS/HLA :APPLICATION A LA MODELISATION ET SIMULATION DISTRIBUEE DE WORKFLOW

Gregory Zacharewicz

UNIVERSITÉ PAUL CÉZANNE AIX-MARSEILLE III

N° d'identification : 2006AIX30053

TITRE :

UN ENVIRONNEMENT G-DEVS/HLA : APPLICATION A LA MODELISATION ET SIMULATION DISTRIBUEE DE WORKFLOW

THÈSE
pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ Paul CÉZANNE AIX-MARSEILLE III
Faculté des Sciences et Techniques

Discipline : Modélisation et Conception des processus Assistés par Ordinateur

Ecole Doctorale en Mathématiques et Informatique de Marseille (ED 184)

Présentée et soutenue publiquement par

Grégory ZACHAREWICZ

Le jeudi 30 novembre 2006

JURY

M. M. ADELANTADO, Maître de recherches (HDR), ONERA-CERT/DPRS Toulouse,
Mme. C. FRYDMAN, Professeur, Université Paul Cézanne Aix Marseille III (Directeur de thèse),
M. N. GIAMBIASI, Professeur, Université Paul Cézanne Aix Marseille III (Directeur de thèse),
M. E. RAMAT, Professeur, Université du Littoral, Côte d'Opale, Calais,
M. G. WAINER, Professeur Associé, Université Carleton, Ottawa, Canada (Rapporteur),
M. B.P. ZEIGLER, Professeur, Université d'Arizona, Tucson, USA (Rapporteur).

ANNEE : 2006

à Yolande, Robert,
Brice,
Paulette.

Remerciements

Cette thèse s'est déroulée au sein de l'équipe COmmande et SImulation du Laboratoire des Science de l'Information et des Systèmes (Unité Mixte de Recherche CNRS 6168) de l'Université Paul Cézanne. Ce travail a été réalisé dans le cadre d'une convention CIFRE de l'ANRT avec la société ITESOFT. Que cette institution et cette société trouvent ici le témoignage de ma reconnaissance.

Je tiens tout particulièrement à exprimer ma plus profonde gratitude à Madame Claudia FRYDMAN et Monsieur Norbert GIAMBIASI, Professeurs à l'Université Paul Cézanne, pour m'avoir fait confiance depuis mon DEA, avoir accepté de diriger ces travaux de thèse et m'avoir soutenu tout au long de cette étude. Leurs conseils ont été indispensables à la concrétisation de cette recherche.

À Monsieur Gabriel WAINER, professeur à l'Université Carleton, Ottawa et Monsieur Bernard P. ZEIGLER, Professeur à l'Université de l'Arizona, j'adresse ma plus respectueuse reconnaissance pour l'intérêt qu'ils ont porté à ce travail en acceptant d'en être les rapporteurs dans ce jury.

Que Monsieur Martin ADELANTADO, Maître de recherches à l'ONERA-CERT/DPRS à Toulouse, et Monsieur Eric RAMAT, Professeur à l'Université du Littoral, à Calais, qui me font l'honneur de participer à ce jury, trouvent ici l'expression de ma profonde gratitude.

Je voudrais également exprimer toute ma reconnaissance à Monsieur Didier CHARPENTIER qui m'a accueilli dans son entreprise et pour avoir accepté de participer à ma soutenance.

Je remercie tous les doctorants, chercheurs et personnels du LSIS pour leur grande gentillesse et leur contribution à l'élaboration de ce travail, avec une pensée particulière à (par ordre d'apparition à l'écran) Gwen, Philippe, Erwan, Alain, Aline, Ronald, Sega, Abderrahmane, Mamadou, François M., Lasaad, Sébastien et Salam et plus généralement aux voisins du premier étage. Ma plus profonde reconnaissance va à Amine, Julien et Olivier qui pendant plus de trois ans, m'ont donné des conseils précieux et beaucoup de leur temps pour élaborer ce travail.

Ma gratitude éternelle va à mes parents, ma grand-mère, mon frère et à toute ma famille (coté fr, sp, pl) pour leur soutien sans faille durant toutes ces années d'études. Un grand merci à mon père, pour m'avoir encouragé sans cesse et avoir relu plusieurs fois ce manuscrit. Finalement, un grand merci à Sophie, François, Carin, Vincent, Steph, Nico, Ali, Benoît et tous ceux dont j'oublie les noms qui m'ont été d'un soutien indispensable dans des moments quelquefois difficiles.

Résumé étendu

Les travaux de cette thèse portent sur :

- la proposition d'améliorations d'algorithmes de simulation distribuée conservative de modèles G-DEVS (Generalised Discrete Event Specification) et DEVS,
- la définition et la réalisation d'un environnement de modélisation & simulation de modèles G-DEVS compatible HLA (High Level Architecture) implémentant les améliorations proposées.
- l'application de l'environnement à la modélisation et la simulation d'un Workflow.

Les fonctions de communication et de synchronisation des informations échangées entre les composants distribués respectent la norme HLA. Dans ce contexte, nous avons proposé un coordinateur racine distribué incluant un algorithme de communication avec le RTI (Run Time Infrastructure) HLA, ce qui permet, entre autres, de conserver la modularité et l'indépendance des modèles par rapport à la méthode de simulation. Cette première amélioration, basée sur le mécanisme de synchronisation conservative, permet, en outre, d'utiliser un Lookahead positif. Nous avons ensuite proposé deux algorithmes originaux pour le calcul d'un Lookahead relatif à l'état courant d'un modèle distribué. Ces algorithmes permettent d'augmenter les performances de la simulation distribuée comme l'illustrent les expériences que nous avons menées. Dans ces approches, les modèles G-DEVS possèdent une fonction de durée de vie des états dépendant d'une seule variable d'état (première solution) ou de plusieurs variables d'état (seconde solution). Ces algorithmes sont basés sur l'analyse du domaine de variation de la fonction de durée de vie impliquée dans la détermination du Lookahead du modèle considéré.

Basé sur ces approches, nous avons développé un environnement de modélisation et simulation distribuée de modèles G-DEVS compatibles HLA. Cet environnement a été intégré à une application de Workflow à laquelle nous avons adjoint un module de transformation automatique de Workflow en modèles G-DEVS. Des applications de cet environnement à des cas réels d'entreprises illustrent les possibilités offertes et les performances de l'approche proposée pour la simulation distribuée.

Mots clés : DEVS, GDEVS, Simulation Distribuée, Synchronisation Conservative, Lookahead, HLA, Gestion du temps HLA, FEDEP HLA, Workflow

Extended Abstract

The works of this PhD thesis are mainly divided in:

- *the proposition of algorithms improvements for conservative distributed simulation of G-DEVS (Generalised Discrete Event Specification) and DEVS models,*
- *the definition and the development of a modeling and simulation environment based on G-DEVS models HLA (High Level Architecture) compliant that implements the proposed improvements,*
- *the application of the environment for modeling and simulation of Workflow.*

The functions that synchronize the data exchange between the distributed components respect the HLA standard. In this context, we proposed a distributed root coordinator including a communication algorithm with the HLA RTI (Run Time Infrastructure), which preserves the modularity and the independence of the models with regard to the method of simulation. This first improvement, based on the conservative synchronization mechanism, use a positive Lookahead value. Then, we proposed two original algorithms for computation of Lookahead depending on the current state of a distributed model. These algorithms increase the performances of the distributed simulation as the experiments, we performed, illustrate it. In these approaches, the G-DEVS model possesses a state lifetime function depending on a single state variable (first solution) or on several state variables (second solution). These algorithms analyze the variation domain of the lifetime function involved in the determination of the considered model Lookahead.

Based on these approaches, we developed an environment for modeling and distributed simulation of G-DEVS models HLA compliant. This environment was integrated into a Workflow application to which we added a module of automatic transformation of Workflow process models in G-DEVS models. Applications of this environment to real cases from industrial world illustrate the offered possibilities and the performances of the approach proposed for distributed simulation.

Key words: DEVS, GDEVS, Distributed Simulation, Conservative Synchronization, Lookahead, HLA, HLA time Management, HLA FEDEP, Workflow

Sommaire

INTRODUCTION GENERALE **1**

CHAPITRE 1. ETAT DE L'ART **5**

1 INTRODUCTION **5**

2 MODELISATION & SIMULATION **5**

2.1 INTRODUCTION A LA MODELISATION ET SIMULATION 5
2.2 LES NIVEAUX DE SPECIFICATION D'UN SYSTEME 6
2.3 LES FORMALISMES DE SPECIFICATION DES SYSTEMES 6
2.3.1 Le Système et son cadre expérimental *7*
2.3.2 Le Modèle *8*
2.3.3 Les Classes de modèles (formalismes) *10*
2.3.4 Le Simulateur *11*
2.4 MODELISATION ET SIMULATION A EVENEMENTS DISCRETS 13
2.4.1 Formalisme DEVS *14*
2.4.2 Formalisme GDEVS *16*
2.4.3 Classe de modèles G-DEVS$_{PF}$ *18*
2.5 CONCLUSION MODELISATION & SIMULATION 20

3 SIMULATION DISTRIBUEE **20**

3.1 SOLUTIONS ARCHITECTURALES DISPONIBLES 20
3.2 METHODES DE DISTRIBUTION POSSIBLES 21
3.3 SIMULATION DISTRIBUEE A EVENEMENTS DISCRETS 21
3.3.1 Evolutions *21*
3.3.2 Exécution distribuée : respect de la causalité *21*
3.3.3 Types de synchronisation *21*
3.4 ARCHITECTURES DE SIMULATION 24
3.4.1 DIS *24*
3.4.2 ALSP *24*
3.4.3 HLA *25*
3.4.4 Techniques de simulation distribuée « non dédiées » *34*
3.5 CONCLUSION SIMULATION DISTRIBUEE 35

4 WORKFLOW **35**

4.1 UNE INTRODUCTION AU WORKFLOW 35
4.2 ORIGINES 36
4.3 MOTIVATIONS 37
4.4 DEFINITIONS ET TERMINOLOGIES 37
4.5 DEFINITIONS DE BASE DU WORKFLOW 38
4.5.1 Définition d'un Workflow *38*
4.5.2 Méta Modèle basique *39*
4.5.3 Concepts et Terminologie Workflow fondamentaux *39*
4.5.4 Concepts secondaires (Wider Workflow Concepts & Terminology) *41*
4.6 DIMENSIONS 43
4.7 DESCRIPTION DES SYSTEMES WORKFLOW 43
4.7.1 Build time *44*
4.7.2 Run time *44*

4.7.3 Utilisateurs, outils et applications ... *45*
4.8 MODELE DE REFERENCE DES SYSTEMES WORKFLOW ... 45
4.8.1 Interface avec les Outils de définition de procédures ... *45*
4.8.2 Interface avec les applications clientes Workflow ... *45*
4.8.3 Interface avec les applications invoquées ... *46*
4.8.4 Interface avec les autres Workflow ... *46*
4.8.5 Interface avec les outils de contrôle et d'administration ... *46*
4.9 CLASSIFICATION DES SYSTEMES WORKFLOW ... 47
4.9.1 Processus collaboratifs ... *47*
4.9.2 Workflow administratif ... *48*
4.9.3 Workflow de production ... *48*
4.9.4 Workflow adaptable ou Workflow ad hoc ... *49*
4.10 MODELISATION DES PROCESSUS WORKFLOW ... 51
4.10.1 Aspects à modéliser ... *51*
4.10.2 Modélisation de Workflow adaptable ... *52*
4.10.3 Techniques et outils de modélisation de Workflow ... *53*
4.11 OUTILS LOGICIELS POUR DECRIRE, ANALYSER ET VALIDER UN WORKFLOW ... 54
4.12 CONCLUSION WORKFLOW ... 54

5 CONCLUSION ... 56

CHAPITRE 2. MODELISATION & SIMULATION G-DEVS / HLA ... 57

1 INTRODUCTION ... 57

2 SIMULATION SEQUENTIELLE DE MODELES G-DEVS ... 58

2.1 SIMULATION DES MODELES DEVS ... 58
2.1.1 Rappel Simulateur conceptuel DEVS ... *58*
2.2 MISE A PLAT DES MODELES DEVS/G-DEVS ... 65
2.2.1 Travaux précédents ... *65*
2.2.2 Nouvelle structure de simulation ... *66*
2.2.3 Spécification de mise à plat de modèles hiérarchiques ... *67*

3 SIMULATION DISTRIBUEE DE MODELES G-DEVS ... 69

3.1 COMPOSANTS DE SIMULATION ... 69
3.1.1 Coordinateur local ... *69*
3.1.2 Simulateur ... *70*
3.1.3 Coordinateur Racine Distribué ... *70*
3.2 ALGORITHMES DE TRAITEMENT DES EVENEMENTS ... 71
3.2.1 Coordinateur Local ... *72*
3.2.2 Simulateur ... *75*
3.2.3 Coordinateur Racine Distribué ... *77*
3.3 DEFINITION DE MODELES COUPLES DISTRIBUES CONFORMES A HLA ... 79
3.3.1 Rappel d'Intégrations DEVS/HLA ... *79*
3.3.2 Création d'une fédération suivant le FEDEP ... *80*
3.3.3 Algorithme de communication avec le RTI ... *83*
3.4 AMELIORATION DU CALCUL DU LOOKAHEAD DANS G-DEVS/HLA ... 86
3.4.1 Application du calcul du Lookahead à la classe G-DEVS$_{PF1}$ ... *86*
3.4.2 Application du calcul du Lookahead à la classe G-DEVS$_{PF2}$ ... *89*

4 PERSPECTIVES ... 96

5 CONCLUSION 96

CHAPITRE 3. ENVIRONNEMENT DE SIMULATION G-DEVS / HLA 97

1 INTRODUCTION 97

2 ENVIRONNEMENTS DE MODELISATION GRAPHIQUE DEVS EXISTANTS 98

2.1 CD++ 98
2.2 JDEVS 98
2.3 ANALYSE 98

3 SPECIFICATION DE L'ENVIRONNEMENT LSIS_DME 98

3.1 CONTRIBUTIONS 99
3.2 SPECIFICATION DE L'EDITEUR GRAPHIQUE LSIS_DME 99
3.2.1 Edition de modèles atomiques 99
3.2.2 Edition de modèles couplés 101
3.3 SPECIFICATION DE L'ENVIRONNEMENT DE SIMULATION LSIS_DME 102
3.3.1 Fonctions de mise à plat de la structure hiérarchique de modèles 102
3.3.2 Algorithmes de simulation 107
3.3.3 Calcul de la complexité algorithmique 114
3.3.4 Mise en œuvre de la version distribuée de l'environnement G-DEVS/HLA 119
3.4 ILLUSTRATIONS DE L'ENVIRONNEMENT DE SIMULATION 120
3.4.1 Exemple Commande Système commandé 120
3.4.2 Exemple de Modèle couplé distribué G-DEVS/HLA 123

4 PERFORMANCES 126

4.1 PERFORMANCES DES ENVIRONNEMENTS DEVSCLUSTER & DDEVSIM ++ 126
4.2 MESURES DE PERFORMANCE REALISEES SUR LE SIMULATEUR G-DEVS « MIS A PLAT » 127
4.2.1 Exemples mis en œuvre 127
4.2.2 Discussion 129
4.2.3 Amélioration de la gestion de listes 130
4.3 MESURE DE PERFORMANCE DE SIMULATIONS G-DEVS / HLA 130
4.3.1 Exemples mis en œuvre 131
4.3.2 Réduction du surcoût en temps de simulation 133

5 PERSPECTIVES 134

6 CONCLUSIONS 134

CHAPITRE 4. APPLICATION A UN ENVIRONNEMENT WORKFLOW G-DEVS / HLA 135

1 INTRODUCTION 135

2 PRESENTATION DU CADRE DE TRAVAIL 135

2.1 RAPPEL DU MODELE WORKFLOW DE REFERENCE 135
2.2 DETAILS DE L'INTERFACE 1 136

3 TRANSFORMATION D'UNE STRUCTURE WORKFLOW EN STRUCTURE G-DEVS .. 137

3.1 PROCESSUS DE TRANSFORMATION WORKFLOW G-DEVS 137
3.2 SPECIFICATION TEXTUELLE D'UN WORKFLOW 138
3.2.1 Nécessité d'une spécification textuelle 138
3.2.2 Représentation d'une Procédure Workflow 138
3.2.3 Définition des éléments basiques liés à une Procédure Workflow 140

3.3 DEFINITION DE MODELES WORKFLOW SIMULABLES DANS LE FORMALISME G-DEVS 148
3.3.1 Choix du formalisme des modèles simulables des Workflow 148
3.3.2 Définition d'une Bibliothèque de modèles G-DEVS pour la définition d'un Workflow 149
3.4 ALGORITHME DE TRANSFORMATION 156
3.4.1 Structure source 157
3.4.2 Structure cible 157
3.4.3 Fonction de transformation 157
3.5 SIMULATION DU MODELE G-DEVS COUPLE OBTENU 160

4 DEFINITION D'UN ENVIRONNEMENT WORKFLOW COMPATIBLE HLA 160

4.1 TRAVAUX PRECEDENTS 160
4.2 MODELE DE REFERENCE WORKFLOW COMPATIBLE HLA 162
4.3 CREATION DES TABLES DE FOM DE LA FEDERATION WORKFLOW GDEVS HLA 163

5 APPLICATION A UN WORKFLOW D'ENTREPRISE 164

5.1 CAHIER DES CHARGES 165
5.2 EXEMPLE DE LA ROUTE H80SH27F ATTACHEE AU PRODUIT FDHB2AAG-03 166
5.2.1 Représentation du modèle à l'aide de l'outil de modélisation graphique LSIS_WME 167
5.2.2 Représentation textuelle de la route H80SH27F 168
5.2.3 Représentation des sous-modèles 169
5.3 TRANSFORMATION DU WORKFLOW EN MODELE G-DEVS 173
5.3.1 Résultats de simulation de la route H80SH27F 174
5.4 DEFINITION DU WORKFLOW COMPATIBLE HLA 174
5.4.1 Création des tables d'un FOM HLA 175
5.4.2 Mécanisme de Publication/ Souscription 178
5.5 CONCLUSION DE LA MODELISATION DE LA ROUTE H80SH27F 179

6 PERSPECTIVES 180

7 CONCLUSION 180

CONCLUSION GENERALE 183

BIBLIOGRAPHIE 185

Index des figures

FIGURE 1 - HIERARCHIE DE SPECIFICATION DES SYSTEMES 6
FIGURE 2 - RELATIONS DE MODELISATION [ZEIGLER 00] 7
FIGURE 3 - DIFFERENTS TYPES DE MODELES 11
FIGURE 4 - SCHEMA EVOLUTION ESPACE ET TEMPS [FUJIMOTO 00] 12
FIGURE 5 - CORRESPONDANCE ENTRE MODELE ET SIMULATEUR 16
FIGURE 6 - EXEMPLES DE TRAJECTOIRES CONSTANTES ET POLYNOMIALES PAR MORCEAU 16
FIGURE 7 - REPRESENTATION GRAPHIQUE D'UN MODELE DEVS ATOMIQUE [SONG 94] 19
FIGURE 8 - REPRESENTATION GRAPHIQUE D'UN MODELE DEVS COUPLE [CARSTEN 94] 20
FIGURE 9 - EXEMPLE D'INTERBLOCAGE [SAMADI 84] 22
FIGURE 10 - VUE HAUT NIVEAU, LOGIQUE D'UNE EXECUTION DE FEDERATION HLA 30
FIGURE 11 - FEDERATION DEVELOPMENT AND EXECUTION PROCESS (FEDEP), VUE DE HAUT NIVEAU. 34
FIGURE 12 - LES SYSTEMES DE GESTION DE WORKFLOW DANS UNE PERSPECTIVE HISTORIQUE 37
FIGURE 13 - ENVIRONNEMENT DU SYSTEME WORKFLOW 38
FIGURE 14 - META MODELE WORKFLOW POUR LA DEFINITION DE PROCESSUS [WFMC11 99] 39
FIGURE 15 - RESEAU DE PROCESSUS 42
FIGURE 16 - DIMENSIONS DES SYSTEMES WORKFLOW 43
FIGURE 17 - CARACTERISTIQUES DU SYSTEME WORKFLOW 44
FIGURE 18 - MODELE DE REFERENCE DES SYSTEMES WORKFLOW 45
FIGURE 19 - DIFFERENTES CLASSES DES SYSTEMES WORKFLOW 47
FIGURE 20 - TABLEAU DES FONCTIONNALITES DE LOGICIELS DE MODELISATION WORKFLOW 55
FIGURE 21 - CLASSES DE MODELE ET SIMULATEUR [ZEIGLER 95] 59
FIGURE 22 - CORRESPONDANCE ENTRE MODELES ET SIMULATEUR HIERARCHIQUE [ESCUDE 00] 60
FIGURE 23 - ALGORITHME DU SIMULATEUR DE MODELE DEVS BASIQUE [ZEIGLER 00] 61
FIGURE 24 - ALGORITHME DU COORDINATEUR DE MODELE DEVS COUPLES [ZEIGLER 00] 63
FIGURE 25 - ALGORITHME DU COORDINATEUR RACINE DU SIMULATEUR ABSTRAIT [ZEIGLER 00] 65
FIGURE 26 - CLASSE DU MODELE COUPLE DEVS DANS LSIS_DME 66
FIGURE 27 - MISE A PLAT ET DISTRIBUTION DE LA STRUCTURE DE SIMULATION HIERARCHIQUE 70
FIGURE 28 - MODELE COMPORTEMENTAL DU COORDINATEUR LOCAL 72
FIGURE 29 - ALGORITHME BOUCLE DE TRAITEMENT DU COORDINATEUR LOCAL 72
FIGURE 30 - ALGORITHME RECEPTION AUTORISATION DU COORDINATEUR LOCAL 73
FIGURE 31 - ALGORITHME RECEPTION XMESSAGE DU COORDINATEUR LOCAL 73
FIGURE 32 - ALGORITHME RECEPTION DMESSAGE DU COORDINATEUR LOCAL 74
FIGURE 33 - ALGORITHMES DE GESTION DU LOOKAHEAD DU COORDINATEUR LOCAL 74
FIGURE 34 - ALGORITHME RECEPTION YMESSAGE DU COORDINATEUR LOCAL 75
FIGURE 35 - MODELE COMPORTEMENTAL DU NIVEAU SIMULATEUR 76
FIGURE 36 - CALCUL DU LOOKAHEAD AU NIVEAU SIMULATEUR 76
FIGURE 37 - ALGORITHME NEXTEVENTREQUEST DU COORDINATEUR RACINE 77
FIGURE 38 - MODELE COMPORTEMENTAL DU COORDINATEUR RACINE 78
FIGURE 39 - INTEGRATION DU SIMULATEUR G-DEVS DISTRIBUE AVEC LE RTI 81
FIGURE 40 - TABLE DE STRUCTURE DES INTERACTIONS 81
FIGURE 41 - TABLE DE PARAMETRES 82
FIGURE 42 - ALGORITHME DE COMMUNICATION AVEC LE RTI 85
FIGURE 43 - ALGORITHME DE CALCUL DU LOOKAHEAD RELATIF A L'ETAT ACTUEL DU MODELE DEVS. 88
FIGURE 44 - EXEMPLE DE LOOKAHEAD RELATIF A LA PHASE ACTUELLE DU MODELE G-DEVS$_{PF1}$ 89
FIGURE 45 - ALGORITHME DE DIJKSTRA POUR LE PARCOURS DU GRAPHE D'UN MODELE G-DEVS$_{PF2}$. 93
FIGURE 46 - EXEMPLE DE LOOKAHEAD RELATIF A L'ETAT ACTUEL DU MODELE G-DEVS$_{PF2}$ 95
FIGURE 47 - EDITION D'UN MODELE ATOMIQUE GRAPHIQUE AVEC LSIS_DME 100
FIGURE 48 - EDITION D'UN MODELE COUPLE GRAPHIQUE AVEC LSIS_DME 101
FIGURE 49 - CLASSES DU MODELE COUPLE DEVS DANS LSIS_DME 104
FIGURE 50 - FONCTION DE MISE À PLAT DES MODÈLES DANS LSIS_DME 105
FIGURE 51 - FONCTION DE TRANSFORMATION DU COUPLAGE DANS LSIS_DME 106
FIGURE 52 - EXEMPLE DE MISE A PLAT DE MODELES 107
FIGURE 53 - CLASSES DU SIMULATEUR DE MODELES COUPLE DEVS DANS LSIS_DME 108
FIGURE 54 - ALGORITHME DU COORDINATEUR 110
FIGURE 55 - ALGORITHME DE BOUCLE PRINCIPALE DE TRAITEMENT DES EVENEMENTS PAR LE COORDINATEUR 111

FIGURE 56 - ALGORITHME DE TRAITEMENT DES MESSAGES RETOUR PAR LE COORDINATEUR 112
FIGURE 57 - ALGORITHME DU SIMULATEUR 113
FIGURE 58 - FONCTION DU MODELE 114
FIGURE 59 - ECHEANCIERS DE L'APPROCHE SIMULATION HIERARCHIQUE 115
FIGURE 60 - ECHEANCIERS DE L'APPROCHE SIMULATION NON HIERARCHIQUE 117
FIGURE 61 - COMPARAISON DES COMPLEXITES 118
FIGURE 62 - DIAGRAMME DE CLASSE DE LOCAL_COORD_DISTRIB 120
FIGURE 63 - MODELE COMMANDE SYST COMMANDE 121
FIGURE 64 - FENETRE DE PLANIFICATION DES EVENEMENTS D'ENTREE 122
FIGURE 65 - FENETRE D'AFFICHAGE DE LA TRACE DE LA SIMULATION 123
FIGURE 66 - EXEMPLE DE MODELES COUPLES DISTRIBUES 123
FIGURE 67 - TABLES DU FOM DU MODELE COMMANDE--SYST. COMMANDE 124
FIGURE 68 - COMMUNICATION ENTRE LES FEDERES AU TRAVERS DU RTI. 125
FIGURE 69 - MODELE A HIERARCHIQUE 128
FIGURE 70 - MODELE B HIERARCHIQUE 128
FIGURE 71 - COMPARAISON DES PERFORMANCES « MISE A PLAT » ET HIERARCHIQUE 129
FIGURE 72 - ARBORESCENCE DE SORTIE DES PERFORMANCES AVEC JRAT 130
FIGURE 73 - MODELES ATOMIQUES G-DEVS 131
FIGURE 74 - MODELE COUPLE A 8 MODELES 131
FIGURE 75 - MODELE COUPLE DISTRIBUE A 8 MODELES G-DEVS FEDERES 132
FIGURE 76 - NOMBRE DE MESSAGES TRAITES PAR FEDERE EN FONCTION DE LA TAILLE DE LA FEDERATION 132
FIGURE 77 - TEMPS D'EXECUTION EN FONCTION DU LOOKAHEAD 133
FIGURE 78 - SURCOUT EN FONCTION DE LA VALEUR DE LOOKAHEAD 133
FIGURE 79 - DIAGRAMME DU MODELE DE REFERENCE WORKFLOW 136
FIGURE 80 - DEPUIS UN EDITEUR DE WORKFLOW VERS UN MODELE G-DEVS 137
FIGURE 81 - BIBLIOTHEQUE DE MODELES WORKFLOW 149
FIGURE 82 - MODELE ATOMIQUE G-DEVS COMPOSANT STOCK 151
FIGURE 83 - MODELE ATOMIQUE G-DEVS COMPOSANT TACHE BASIQUE 151
FIGURE 84 - MODELE ATOMIQUE G-DEVS COMPOSANT RESSOURCE 152
FIGURE 85 - MODELE ATOMIQUE G-DEVS COMPOSANT RESSOURCE AVEC COMPORTEMENT 153
FIGURE 86 - MODELE G-DEVS DU AND-SPLIT 153
FIGURE 87 - MODELE G-DEVS DU AND-JOIN 154
FIGURE 88 - MODELE G-DEVS DU OR/XOR SPLIT 154
FIGURE 89 - MODELE G-DEVS DU OR-JOIN AVEC FUSION 155
FIGURE 90 - MODELE COUPLE G-DEVS D'UN WORKFLOW 156
FIGURE 91 – ALGORITHME DE TRANSFORMATION WF - G-DEVS 159
FIGURE 92 - RESEAU DE PETRI ASSOCIE AVEC COMMUNICATION AVEC LE RTI 161
FIGURE 93 - ENVIRONNEMENT DE GESTION DE FLUX COMPATIBLE « HLA » 161
FIGURE 94 - MODELE DE REFERENCE WORKFLOW COMPATIBLE HLA 163
FIGURE 95 - TABLE D'OBJET DU FOM D'UN WORKFLOW 163
FIGURE 96 - ROUTE GENERIQUE STMICROELECTRONICS 166
FIGURE 97 - ROUTE H80SH27F ATTACHEE AU PRODUIT FDHB2AAG-03 167
FIGURE 98 - MODELE WORKFLOW DE LA ROUTE H80SH27F ST MICROELECTRONICS 168
FIGURE 99 - MODELE WORKFLOW DE LA ROUTE DE REWORK RWKDFCL1R 169
FIGURE 100 - MODÈLE WORKFLOW DU SCRIPT OF MARKL 03 (OPÉRATION 1010 LASER MARKING) 170
FIGURE 101 - MODÈLE WORKFLOW DE SCRIPT OF CLEAN 06 (OPÉRATION 1015) 171
FIGURE 102 - MODELE WORKFLOW DU SCRIPT_XXX (OPERATION 1020) 172
FIGURE 103 - MODELE G-DEVS COUPLE DE LA ROUTE H80SH27F SUR LSIS_DME 173
FIGURE 104 - RESULTAT G-DEVS COUPLE DE LA ROUTE H80SH27F SUR LSIS_DME 174
FIGURE 105 - MODELE REFERENCE DE L'ENVIRONNEMENT WORKFLOW AVEC LES INTERFACES 175
FIGURE 106 - TABLE D'OBJET HLA DE LA FEDERATION ENVIRONNEMENT WORKFLOW 176
FIGURE 107 - TABLES D'ATTRIBUTS HLA DE L'ENVIRONNEMENT WORKFLOW 177
FIGURE 108 - MAQUETTE DES INTERFACES GRAPHIQUES DE MONITORING 178
FIGURE 109 - MAQUETTE DES INTERFACES GRAPHIQUES DES APPLICATIONS 179

Introduction générale

La répartition planétaire des entreprises d'une part et l'expansion rapide des réseaux informatiques, en particulier Internet, d'autre part, ont conjointement mené au développement d'applications parallèles et distribuées. De fait, les applications distribuées abondent pour l'analyse de problèmes notamment dans le domaine des transports, pour la définition de mondes virtuels, pour l'entraînement militaire et civil et pour la gestion de processus industriels. Ces applications requièrent toutes des capacités de flexibilité et de répartitions des moyens, en conservant des objectifs de performances et d'efficacité. Le développement de protocoles de simulation distribuée de structures hiérarchiques de modèles répond à ces objectifs.

La modélisation et la simulation à événements discrets permettent de traiter des problèmes concernant des systèmes dont le comportement temporel est trop complexe pour être traité analytiquement. La spécification formelle de ces modèles peut être établie grâce au formalisme DEVS [ZEIGLER 76]. DEVS offre, à la différence d'autres formalismes de modélisation & simulation (M&S), une plus grande généralité. Enfin, la Généralisation de DEVS : G-DEVS [GIAMBIASI 98] propose des abstractions à événements discrets de systèmes à trajectoires d'entrée-sortie polynomiales par morceaux au lieu de segments constants par morceaux pour les modèles à événements discrets classiques.

Certaines applications (pour des raisons de performances, de confidentialité,…) requièrent que les différents composants d'un modèle soient situés sur plusieurs ordinateurs avec, éventuellement, des systèmes d'exploitation hétérogènes. La mise en œuvre d'une simulation distribuée de ces modèles nécessite alors un protocole de communication permettant l'échange d'information entre les différents composants. L'apparition de standards de spécification de simulation distribuée de haut niveau (notamment HLA) a permis de faciliter la mise en œuvre de telles simulations.

Les travaux présentés dans ce mémoire avaient pour principaux objectifs de proposer des algorithmes de simulation distribuée conservative de modèles DEVS / G-DEVS, de définir et réaliser un environnement de M&S G-DEVS compatible HLA implémentant les algorithmes proposés, et enfin, d'appliquer l'environnement à la M&S de Workflow.

Dans ce cadre, nous proposons un environnement de simulation distribuée à événements discrets basés sur le formalisme G-DEVS et sur une interface logicielle compatible avec la norme HLA. Nous avons défini et implémenté de nouveaux algorithmes d'intégration et de communication qui utilise le « Lookahead d'HLA » afin, entre autres, d'améliorer les performances de la simulation distribuée.

Les Workflow permettent la gestion assistée par ordinateur des tâches composant un processus administratif ou industriel [COURTOIS 96]. L'application de l'environnement G-DEVS / HLA permet de fournir un cadre formel pour la modélisation et la simulation distribuée des Workflow. Les modèles Workflow générés pourront être ainsi vérifiés, réutilisés et exécutés de façon distribuée grâce à leur compatibilité HLA. En retour, le cadre applicatif des Workflow de production permettra de valider l'environnement de M&S distribuée G-DEVS / HLA défini.

Après un bref état de l'art sur les principaux concepts de Modélisation et de Simulation, nous rappelons, dans le premier chapitre, les concepts de bases des formalismes DEVS et G-DEVS pour la spécification de modèles à événements discrets. Nous présentons ensuite les principes généraux de la simulation distribuée et notamment la norme HLA. Afin d'introduire le cadre applicatif de cette thèse, nous terminons ce premier chapitre par un rappel sur les systèmes de gestion des procédures industrielles nommés Workflow.

Le deuxième chapitre est consacré à la spécification d'un environnement de simulation distribuée de modèles G-DEVS. Dans une première partie, nous définissons un simulateur de modèles G-DEVS possédant une fonction de mise à plat des modèles. Cette fonction est utilisée préalablement à la simulation dans l'objectif d'un gain de performance. Dans une deuxième partie, nous définissons les primitives permettant d'assurer la communication, au sein d'une simulation globale distribuée, entre des simulateurs G-DEVS repartis. Nous montrons que la norme HLA permet de mettre en œuvre les primitives précédentes en proposant des méthodes d'échanges de messages synchronisés entre les simulations locales. De plus, nous présentons un nouvel algorithme de communication G-DEVS/HLA utilisant un Lookahead HLA positif. Enfin, nous introduisons deux méthodes de calcul du Lookahead prenant en considération le comportement des modèles G-DEVS. La première méthode considère des modèles G-DEVS possédant une fonction durée de vie des états dépendante d'une seule variable d'état, la seconde méthode considère des durées de vie calculées à partir de plusieurs variables d'état. Ces algorithmes sont basés sur l'analyse du domaine de variation de la fonction durée de vie impliquée dans la détermination du Lookahead du modèle considéré.

Le chapitre 3 présente la mise en œuvre de l'environnement de modélisation et simulation G-DEVS/HLA que nous proposons. Cet environnement est basé sur la structure de modèles et sur le simulateur abstrait définis par B.P. Zeigler. Cependant, certaines spécificités, pour la

définition graphique de modèles G-DEVS et pour la mise à plat des modèles, sont ajoutées. En outre, nous montrons que le nouveau moteur de simulation « mis à plat » possède un algorithme moins complexe que celui du moteur hiérarchique et réduit ainsi la durée de simulation. Enfin l'environnement comporte une extension permettant de définir des modèles G-DEVS en tant que fédérés HLA. Cette extension de l'environnement permet ainsi de simuler des modèles G-DEVS distribués. Dans ce cas, tous les sous-modèles distribués intercommuniquent en échangeant des données au travers d'un réseau local ou par Internet, pour ce faire nous nous conformons à la norme HLA et utilisons le RTI associé à cette norme pour orchestrer l'échange de données.

Le dernier chapitre présente une application de l'environnement G-DEVS/HLA permettant de modéliser et simuler les systèmes Workflow. Nous montrons que la structure des procédures d'un Workflow peut être aisément validée et modifiée grâce à la modularité des modèles G-DEVS et que l'interopérabilité avec les autres applications, les ressources et les autres systèmes du Workflow, peut être assurée par le respect de la norme HLA. Nous définissons la structure des éléments basiques d'un Workflow, donnons une bibliothèque des composants Workflow basiques réutilisables représentés par des modèles atomiques G-DEVS et nous présentons un algorithme de transformation automatique des modèles Workflow en modèle G-DEVS. De plus, nous proposons une version distribuée (compatible HLA) du « modèle de référence » Workflow [WFMC11 99]. Finalement, nous illustrons l'utilisation de l'environnement G-DEVS/HLA par un exemple de modélisation d'un processus Workflow de fabrication de Wafer électronique.

Chapitre 1. Etat de l'art

1 Introduction

Dans ce chapitre nous rappelons les principaux concepts de la Modélisation et de la Simulation. Nous développons, en particulier, la modélisation à événements discrets et la simulation distribuée de cette classe de modèles. Enfin, nous présenterons les définitions inhérentes aux systèmes de gestion des procédures appelés Workflow afin d'introduire le cadre applicatif de cette thèse.

2 Modélisation & Simulation

On ne résout pas un problème directement sur un système *réel*, mais toujours au travers d'un modèle que nous construisons de ce système. « Un modèle *M* d'un système *S* pour une expérimentation *E* est toute chose à laquelle on peut appliquer *E* pour répondre à des questions concernant *S* » [MINSKY 56]. Un modèle est une forme intelligible d'un système construit pour permettre de trouver une réponse à un problème précis. En agissant sur les modèles à partir de jeux de données et d'une base de temps on définit une simulation.

2.1 Introduction à la Modélisation et Simulation

La théorie de la modélisation et simulation a été introduite dans les années 70. Bernard P. Zeigler a proposé, à cette époque, une décomposition de la théorie de la modélisation selon deux aspects orthogonaux [ZEIGLER 76] :

Les **niveaux de spécification** sont les niveaux qui définissent le comportement du système et les mécanismes qui permettent d'exprimer sa dynamique. Ces niveaux font référence à ses propres travaux ainsi qu'à ceux de George Klir qui définit des niveaux de connaissance des systèmes [KLIR 85].

Les **formalismes de spécification** des systèmes définissent des classes de modèles. Un modèle est construit en respectant un paradigme. Un paradigme est un ensemble de concepts, de lois et de moyens visant à définir une collection de modèles.

2.2 Les niveaux de spécification d'un système

« Un système est une source potentielle de données » pour B.P. Zeigler, mais il reste à sélectionner les données pertinentes et à les observer. Pour cela, [KLIR 85] introduit une représentation de la spécification d'un système décomposée en niveau de connaissance.

Le **niveau Source 0** de la représentation de [KLIR 85] identifie la partie du monde réel que nous souhaitons modéliser et les moyens par lesquels nous allons l'observer.

Le **niveau Donnée 1** définit une base de données de mesures et d'observations faites pour le système source.

Le **niveau Génératif 2**, donne la capacité de recréer les données précédentes utilisant une représentation plus compacte sous forme de formule. Le niveau génératif constitue la connaissance que nous n'avions pas au niveau du système de données.

Enfin, le **niveau Structure 3** indique comment produire des données en termes de composantes systèmes inter connectées.

Parallèlement, B.P. Zeigler [ZEIGLER 00] a proposé une représentation des niveaux de spécification (cf. Figure 1) assez proche de celle de G. Klir mais plus orienté sur la modélisation et simulation.

Niveau	Nom de Spécification	Correspondance avec Klir	Ce qui est connu à ce niveau
0	Cadre d'observation	Système source	Comment stimuler le système avec des entrées. Quelles variables mesurer et comment les observer au travers d'une base de temps
1	Comportement E/S	Données	Les données collectées depuis un système source indexées dans le temps, elles consistent en des couples entrées sorties
2	Fonction E/S		L'état initial. A partir d'un état initial, chaque stimulus d'entrée produit une sortie unique
3	Transition d'état	Génératif	Comment les états sont affectés par les entrées. A partir d'un état et d'une entrée quel est l'état suivant après que le stimulus d'entrée soit reçu ; quel événement de sortie est généré par l'état
4	Couplage de composants	Structure	Les composants et comment ils sont couplés entre eux. Les composants peuvent être spécifiés à un niveau inférieur ou être eux-mêmes des systèmes structurés construits hiérarchiquement

Figure 1 - Hiérarchie de spécification des systèmes

2.3 Les formalismes de spécification des systèmes

Il existe deux classes de méthodes de résolution de problèmes. Les méthodes analytiques et les méthodes basées sur la simulation. Les premières permettent de rechercher une solution générale mais se heurtent au problème de la complexité du modèle. Les méthodes basées sur la simulation permettent de traiter des modèles plus complexes, mais se limitent à des solutions particulières.

B.P. Zeigler [ZEIGLER 00] présente une structure pour la M&S[1] (cf. Figure 2), il y définit les entités essentielles et leurs rapports primordiaux dans la définition d'une M&S. En effet, les termes « modèle » et « simulateur » sont souvent utilisés dans la pratique actuelle sans toujours un réel fondement ; cette structure identifie les questions de base et les problèmes rencontrés dans l'exécution des activités de M&S, ils peuvent ainsi être mieux compris et générer des solutions plus cohérentes. Il est donc important de comprendre ce qui est inclus et exclus des définitions.

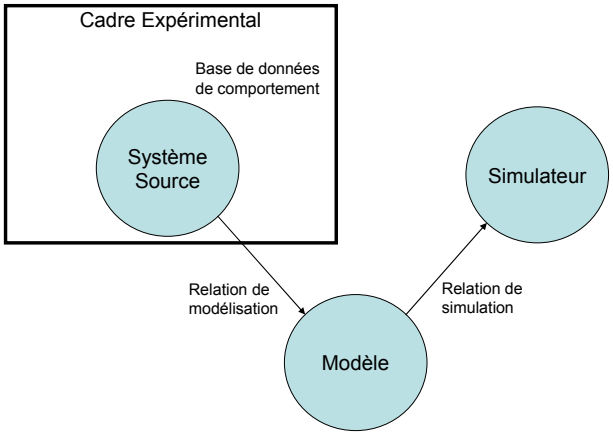


Figure 2 - Relations de modélisation [ZEIGLER 00]

Les entités de la structure sont : le *système source***,** le *cadre expérimental*, le *modèle,* et le *simulateur* ; ils sont reliés par les relations de *modélisation* et de *simulation*. Chaque entité est formellement caractérisée comme un système à un niveau approprié de spécification d'un système dynamique générique. De même chaque relation est caractérisée comme un morphisme approprié entre deux systèmes dynamiques. La présentation exhaustive de la théorie des systèmes dépasse le cadre de ce mémoire mais pourra être trouvée dans [ZEIGLER 00].

2.3.1 Le Système et son cadre expérimental

Le *système source* est la partie délimitée (niveau 0 de spécification) du monde réel ou virtuel que nous souhaitons représenter. Il est considéré comme une boite noire sur lequel l'environnement agit et qui réagit à ces stimuli. En d'autres termes, il est vu comme une *source de données observables*, sous formes de trajectoires des variables indexées par le temps. Les données qui sont recueillies de l'observation ou en expérimentant le système sont appelées *base de données de comportement de système* (niveau 1 de spécification). Il faut se rappeler que ces données sont vues ou générées lorsque le système est plongé dans un contexte particulier du monde réel, et pour prendre en considération cela, [ZEIGLER 00] identifie ce contexte par des cadres expérimentaux utiles pour le modélisateur.

Un *cadre expérimental* est une spécification des conditions dans lesquelles le système est observé ou dans lesquelles on l'expérimente. Un cadre expérimental est la formulation opéra-

[1] Modélisation et Simulation

tionnelle des conditions dans lesquelles se déroule un projet de modélisation et de simulation. Un cadre est définit comme un système qui interagit avec le système étudié pour obtenir les données étudiées dans les conditions spécifiées. Dans certains processus de M&S, la prise en considération du cadre expérimental est omise ou mal délimitée impliquant ainsi des omissions dans les modèles crées et donc des imprécisions dans les résultats obtenus par simulation.

2.3.2 Le Modèle

Il existe de multiples significations attribuées au mot « modèle ». Un modèle peut être conçu à partir d'une représentation physique, mathématique, ou logique afin de reproduire un système, une entité, un phénomène, ou un processus. La définition, en terme de spécification de système, détient pour avantages de reposer sur une base mathématique formelle et se base sur une sémantique clairement définie que chacun peut comprendre de façon non équivoque.

D'un point de vue général, un *modèle* est une spécification d'un système réel ou virtuel. Dans le contexte de la M&S, la spécification de système est classiquement décrite à un haut niveau. Cette spécification peut être décrite par un jeu d'instructions, de règles, d'équations, ou de contraintes pour produire un comportement d'entrée/sortie, ce comportement a pour objectif de produire des données de valeurs proches des données observées sur le système réel.

La relation de modélisation entre le système réel et le modèle définit les parties du système à représenter et comment elles le seront. Plus en détail, nous construisons un modèle de génération de sorties acceptant des trajectoires d'entrée et produisant en réponse des trajectoires de sortie. A ce stade, il apparaît que certaines entrées observées ne produisent pas toujours les mêmes sorties. En effet, la spécification de niveau 1 est non déterministe (ambiguë ou incomplète). Il est donc nécessaire d'introduire la notion « d'état initial » (niveau 2 de spécification) pour trouver une correspondance unique entre les entrées et sorties en fonction d'un état initial du modèle. Ce niveau vise à générer un modèle conceptuel implémentable sur un calculateur à la condition de savoir revenir dans l'état initial [GIAMBIASI 01b]. Il faut ensuite décrire les mécanismes de transition d'état (niveau 3 de spécification) et la prise en compte de l'état courant pour la génération de sorties.

Par ailleurs, la construction de modèles suppose des hypothèses simplificatrices provenant de plusieurs facteurs : choix du modélisateur, connaissance partielle du système, limitation de l'outil de modélisation. Ces hypothèses permettent de garder à l'esprit la différence entre le système et modèle.

Au niveau du modèle il convient de distinguer le modèle conceptuel et le modèle informatique. Le modèle informatique doit être implémentable sur un calculateur et doit donc respecter certaines règles. A partir des niveaux 3 et 4 de la spécification, il est possible de définir directement des spécifications exécutables si l'on traduit la spécification dans un formalisme avec une sémantique opérationnelle. Plus en détail, les modèles conceptuels doivent respecter

les deux principes suivants pour être également considérés comme modèles informatiques [GIAMBIASI 95] :

Le **principe de causalité** définit que le futur ne peut influencer le passé. L'état du système à l'instant présent (t) est indépendant de tout ce qui peut se produire à toute date (t') supérieure à (t).

Le **principe de déterminisme** définit que le futur du système peut être déterminé à partir de son état présent et de son passé. A tout instant (t), il existe une valeur positive (e) telle que le comportement du système puisse être calculé jusqu'à (t + e).

Le modèle représente un point de vue de l'entité modélisée. Ce point de vue est fonction du modélisateur, cependant, la théorie des systèmes distingue classiquement deux points de vue : comportemental et structurel.

2.3.2.1 Le modèle comportemental :

Ce modèle est caractérisé par les variables, le temps et les fonctions qui expriment des relations entre les variables et le temps.

L'état du modèle est définit par un sous ensemble nécessaire des variables pour connaître le prochain état. Ce sous ensemble est nommé ensemble des variables d'état.

Une variable d'états est une variable descriptive nécessaire à un instant t pour calculer la valeur future d'au moins une des variables descriptive du modèle. [GIAMBIASI 01b]

Le comportement du modèle est défini par des fonctions qui expriment le moyen de passer d'état en état. Le modèle comportemental contient donc les règles pour faire évoluer le modèle d'un état à un autre état, en d'autres termes sa dynamique.

De tels modèles forment les composants de base de modèles plus complexes (niveau 4 de spécification) qui sont construits par couplage de modèles entre eux pour former des modèles à spécification de niveau supérieur : les modèles structurels.

2.3.2.2 Le modèle structurel :

Ce modèle définit la structure du modèle, en d'autres termes, quels sont les sous-modèles interconnectés qui définissent le comportement global du modèle. Un concept important est la décomposition, il définit comment le système peut être décomposé en sous-composants systèmes. Un autre concept est la composition qui précise comment des modèles peuvent être couplés pour former des modèles plus grands. Ces concepts reposent sur la propriété nommée « fermeture sous la composition » [ZEIGLER 99] issue de la théorie des systèmes, cette théorie repose sur la preuve mathématique de l'équivalence d'un modèle comportemental et structurel. Les deux types de modèles peuvent être exprimés dans les mêmes termes de la théorie des systèmes. Il est ainsi possible d'obtenir un modèle ayant une construction hiérarchique composé de sous-modèles modulaires qui possèdent des portes d'entrée et de sortie interconnectées et qui communiquent afin de reproduire un comportement global.

2.3.3 Les Classes de modèles (formalismes)

Les modèles peuvent être exprimés dans une variété de formalismes qui peuvent être compris comme des moyens pour spécifier les sous-classes de systèmes dynamiques. Ces formalismes sont choisis en fonction de l'objectif de l'étude à réaliser. Les modèles formels, reposant sur des concepts et propriétés mathématiques, possèdent deux dimensions : l'espace et le temps [OREN 87].

2.3.3.1 Classification temporelle

La représentation du temps peut être continue ou discrète.

Modèle à temps continu : la variable qui représente le temps prend des valeurs réelles.

Modèle à temps discret : le temps progresse par pas temporel, les valeurs prises de la variable temps sont des multiples du pas.

2.3.3.2 Classification spatiale

La deuxième dimension de classification des modèles est l'espace dont les variables descriptives peuvent prendre des valeurs dans un ensemble fini ou infini.

Modèle à variables continues : les variables descriptives prennent des valeurs réelles.

Modèle à variables discrètes : l'ensemble des valeurs des variables est fini.

2.3.3.3 Formalismes de spécification

Les différents formalismes pour représenter un système résultent de la combinaison des classifications de l'espace et du temps présentées ci-dessus. La Figure 3, proposée dans [GIAMBIASI 01b], présente ces formalismes.

.1 Formalisme à équations différentielles : Association d'une base de temps et de variables également continues, les équations définissent les changements d'états.

.2 Formalisme à temps discret : Les variables sont ici continues, l'évolution est définie par pas temporels discrets. Ce formalisme est également nommé équation aux différences.

.3 Formalisme à événements discrets : Association d'une base de temps continue et de changements d'états discrets. Les changements de valeurs des variables, appelés « **événements** », se produisent à des instants nommés « **date d'occurrence** » d'événements, ces dates sont des nombres réels non négatifs. L'évolution de l'état du modèle survient suite au traitement d'un événement à un instant quelconque.

.4 Formalisme à états discrets et temps discret : Association d'une base de temps discrète et de variables d'état discrètes, également nommé machine à états finis synchrones.

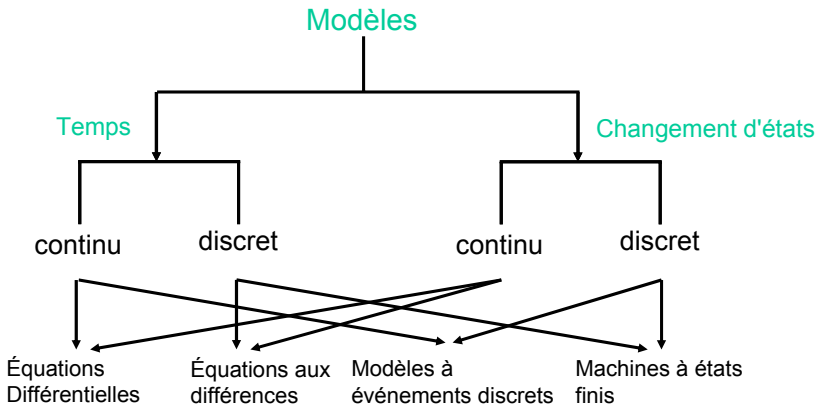


Figure 3 - Différents types de modèles

Il faut garder à l'esprit que la simulation sur calculateur digital implique la discrétisation du temps, de l'espace ou des deux. Certains formalismes présentés ci-dessus possèdent déjà une composante discrète, dans le cas contraire, il faudra réaliser la discrétisation du temps ou de l'espace. Pour les modèles décrits par des équations différentielles, le temps est discrétisé dans les techniques d'intégrations numériques.

2.3.4 Le Simulateur

Un *simulateur* est un système de calcul qui obéit à des instructions pour exécuter un modèle et en produire son comportement [ZEIGLER 00]. Ce dispositif technique peut être, par exemple, un ordinateur mono processeur, un réseau de processeurs, ou plus abstraitement un processeur logique ou un algorithme.

2.3.4.1 La relation de simulation

La relation de simulation permet de vérifier qu'en obéissant aux instructions définies dans un modèle, le simulateur reproduit un comportement spécifié. C'est-à-dire que le simulateur génère correctement des sorties en fonction d'un état initial, de données d'entrée et du temps.

2.3.4.2 La notion de temps

Il existe plusieurs notions importantes relatives au « temps » lorsque l'on parle de simulation. Il est très important de différencier ces notions, en effet la confusion de celles-ci est une source d'erreur courante en simulation. Pour remédier à cela [FUJIMOTO 00] présente un ensemble de définitions relatives au temps :

Temps physique : ce temps fait référence au temps dans le système physique.

Temps simulé : est la représentation du temps physique pendant la simulation, c'est une abstraction. Le temps simulé est défini par un ensemble de valeurs complètement ordonnées ou chaque valeur représente un instant du système physique modélisé.

Temps absolu (horloge) : temps réel qui s'écoule pendant l'exécution de la simulation (fourni par une horloge matérielle), une simulation pourra lire ce temps donné par le processeur de l'ordinateur sur lequel se déroule la simulation.

Chacun des concepts de temps utilise une échelle ou un axe du temps sur lequel sont représentées les notions de précédence entre deux instants. Par exemple si (t_1) et (t_2) représentent deux dates simulées et si ($t_1 < t_2$), alors nous pouvons dire que (t_1) arrive avant (t_2) et (t_2) arrive après (t_1).

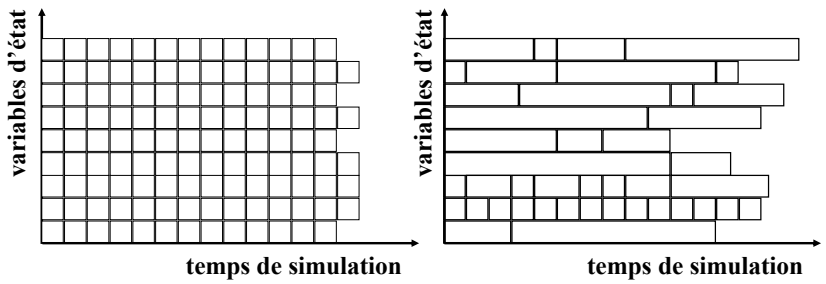
Le temps simulé peut avancer plus lentement que le temps physique si le modèle est complexe à simuler. Le temps peut, dans le cas de modèle plus simple, avancer plus rapidement que le temps physique. Dans ce dernier cas le temps peut avancer aussi rapidement que le temps physique, on parle alors de simulation « temps réel ». Il faut prendre garde dans le cas de la simulation temps réel, le temps simulé avançant de façon inégale, certaines phases de simulation peuvent être associées à des modèles simples dont l'exécution pourra être plus rapide que le temps absolu écoulé. Inversement, certaines phases de simulation peuvent être plus lentes que le temps absolu, dans ce cas la simulation temps réel n'est pas possible car le temps de simulation doit être à tout instant supérieur ou égal au temps physique.

2.3.4.3 Principe de simulation « continue »

La simulation « continue » sur des équations différentielles permet de résoudre des problèmes dont la complexité exclue une solution analytique. La limitation de ces simulations provient du taux d'échantillonnage important nécessaire pour obtenir des solutions non divergentes qui engendre un temps de calcul très important.

2.3.4.4 Principe de simulation discrète

Dans une simulation « discrète », le modèle change d'état en des points discrets sur l'axe des temps simulés. Le programme de simulation met en œuvre l'ensemble des variables d'état et des règles défini dans le modèle permettant de modifier les valeurs des variables au travers du temps de simulation. Les deux catégories de changement d'état les plus communes sont nommées « dirigées par le temps » et « dirigée par les événements ». Ces deux catégories se différencient par le mécanisme d'avancement du temps présentés Figure 4.



(a) dirigée par le temps **(b) dirigée par les événements**

Figure 4 - Schéma évolution espace et temps [FUJIMOTO 00]

2.3.4.5 Evolution dirigée par le temps

Dans une simulation dirigée par le temps (ou synchrone), le temps de simulation est une séquence de **pas** de tailles égales (Figure 4 a). La simulation est cadencée par une horloge

globale qui dirige l'évolution de pas en pas. Dans ce cas, l'algorithme traite un ensemble d'actions associées à un processeur logique pour la plage de temps comprise entre la dernière date traitement et le pas. Notons qu'il peut y avoir zéro ou plusieurs actions associées à la plage de temps. Les actions traitées pour une même date sont supposées indépendantes, il est en effet difficile de déterminer la causalité des actions simultanées dans ce cas. Pour ces raisons le choix de la valeur du pas de temps est primordial dans ce type de simulation, un pas trop petit peut entraîner la simulation d'un certain nombre de dates associées à aucune action. Par opposition un pas trop grand peut poser des problèmes dans la représentation de phénomènes causaux car les actions d'une même plage sont toutes simulées en supposant leurs occurrences simultanées. Une action « cause » survenu pendant un pas donné doit donc, dans ce cas, attendre le prochain pas pour provoquer une action en « conséquence ».

2.3.4.6 Evolution dirigée par les événements

Une autre solution de simulation consiste à ne pas simuler les dates qui sont associées à aucun évènement, on ne simule que lorsque « quelque chose se produit » (l'occurrence d'un événement, cf. Figure 4 b). Ces simulations sont appelées « dirigées par les événements » ou « asynchrones ». Dans ce cas il n'y a pas d'horloge globale, le temps de chaque simulateur avance par saut de durée variable, d'événement en événement à traiter. Ces événements sont stockés par ordre croissant de date d'occurrence dans une liste appelée **échéancier**. Il n'existe donc plus de problème de représentation des événements puisque chaque événement est simulé à sa date exacte et non regroupé sur une date (multiple du pas) comme dans l'évolution dirigée par le temps. L'échéancier est géré par des algorithmes de synchronisation permettant sa mise à jour en cohérence avec l'environnement. Ces algorithmes et échéanciers peuvent être de type séquentiel ou repartis sur différentes machines qui posséderaient une partie de la simulation. Les limitations de ce type de simulation peuvent être liées à la performance dans le cas d'un nombre d'événements important car les événements ne sont pas regroupés et traités « en paquet ». L'autre difficulté est la communication avec un environnement temps réel, dans ce cas il est préférable d'utiliser une évolution dirigée par le temps ou de prévoir des barrières de synchronisation, ces barrières sont des dates qui permettent de se mettre en phase avec l'horloge absolue et les événements provenant du système. Enfin, Il est à retenir que le temps des simulations dirigées par les événements prend ses valeurs dans un ensemble continu, contrairement à l'idée communément reçue.

2.4 Modélisation et simulation à événements discrets

La modélisation à événements discrets requiert deux principes [NUTARO 03]. Le premier est la description non ambiguë du système. Le deuxième est la définition d'algorithmes de simulation à événements discrets dont la validité est fondée et vérifiable.

2.4.1 Formalisme DEVS

B.P. Zeigler définit, dans [ZEIGLER 76], une spécification formelle des modèles à événements discrets : DEVS[2]. Ce formalisme a été introduit comme un formalisme abstrait universel indépendant de l'implémentation. Il peut, par sa capacité d'abstraction, exprimer les systèmes définis dans des formalismes traditionnels comme les équations différentielles (temps continu) et aux différences (temps discret). Le concept de modèles atomiques et couplés, introduit par [ZEIGLER 84], fournit un moyen de construire des modèles composés, en réutilisant des descriptions stockées en librairie.

2.4.1.1 Modèle atomique

Le modèle atomique est un élément non décomposable. Le comportement de cet élémént est régi par un modèle à événements discrets.

Essentiellement, un modèle atomique possède une base de temps, des entrées, des états, des sorties et des fonctions pour déterminer les états suivants et les sorties à partir des états actuels et des entrées. Les événements d'entrées et sorties admissibles sont les seuils de trajectoires continues par morceaux. Les signaux d'entrée sont abstraits avec une trajectoire constante par morceau dont les seuils sont considérés comme des événements discrets.
Formellement, un modèle atomique M est spécifié par un septuplet :

$$M = < X, S, Y, \delta_{int} , \delta_{ext} , \lambda, D>$$

X : ensemble des types des événements externes,

S : ensemble des états séquentiels,

Y : ensemble des types d'événements de sortie

δ_{int} : $\mathbf{S \rightarrow S}$ fonction de transition interne définissant les changements d'état dus à des événements internes,

δ_{ext} : $\mathbf{ST \times X \rightarrow S}$ fonction de transition externe définissant les changements d'état dus à des événements externes,

L'ensemble **ST** des états totaux du système est :

ST = $\{(s, e)\ s \in S, 0 \leq e \leq D\ (s)\}$ où **e** représente le temps écoulé dans l'état s,

λ : $\mathbf{S \rightarrow Y}$ fonction de sortie,

D : $\mathbf{S \rightarrow \mathbb{R}+ \cup \infty}$ fonction durée de vie des états (ou ta[3]).

2.4.1.2 Modèle couplé

Un modèle couplé est un modèle structurel. Il décrit une structure par interconnexion de modèles de base. Chaque modèle de base du modèle couplé interagit avec d'autres modèles pour produire le comportement global. Les modèles de base sont soit des modèles atomiques soit d'autres modèles couplés, le couplage de ces modèles se réalise de façon hiérarchique comme indiqué en Figure 5 a).

---

[2] Discrete Event System Specification

[3] Time advance function

Un modèle couplé à événements discrets est défini par la structure suivante :

$$MC = < X, Y, D, \{M_d / d \in D\}, EIC, EOC, IC, Select >$$

X : ensemble des événements externes.

Y : ensemble des événements de sortie.

D : ensemble des noms de composants.

M_d : modèle DEVS.

EIC : couplages d'entrées externes.

EOC : couplages de sorties externes.

IC : couplages internes.

Select : définit une priorité entre événements simultanés destinés à des composants différents.

2.4.1.3 Simulateur de Modèles couplés DEVS

Parallèlement à l'élaboration des différents modèles DEVS présentés précédemment, B.P. Zeigler a développé le concept de simulateur abstrait [ZEIGLER 00]. L'architecture de simulation est dérivée de la structure hiérarchique des modèles couplés DEVS. Un simulateur abstrait représente une description algorithmique permettant de mettre en œuvre les fonctions du modèle, afin de générer son comportement. Un tel simulateur est obtenu en faisant correspondre à chaque élément du modèle un composant du simulateur. La construction d'un simulateur indépendant du modèle permet une séparation, au niveau réalisation, des parties modélisation et simulation.

Pour effectuer une simulation, une hiérarchie de processeurs, équivalente à la hiérarchie des modèles, est construite. A chaque **composant** du modèle (Figure 5 a) est associé un **processeur** de la structure hiérarchique du simulateur (Figure 5 b). Chaque processeur participe à la simulation en exécutant les fonctions qui expriment la dynamique du modèle. Les processeurs sont :

- ***le Simulateur*** qui assure la simulation des modèles atomiques en utilisant les fonctions définies par DEVS,
- ***le Coordinateur*** qui assure le routage des messages entre les modèles couplés en fonction des définitions de couplage,
- ***le Coordinateur Racine*** qui assure la gestion globale de la simulation. Il ordonne le début et la fin de la simulation et gère l'horloge globale.

La simulation s'effectue grâce à l'échange de messages spécifiques [ZEIGLER 00] entre les différents processeurs comme décrit dans la Figure 5 b). Les trois premiers types de messages ci-dessous représentent les différents événements définis dans DEVS.

- **Xmessage** : utilisé lorsqu'un événement externe arrive sur un modèle,
- ***message** : utilisé pour simuler un événement interne,
- **Ymessage** : utilisé pour simuler un événement de sortie,
- **Imessage :** utilisé pour initialiser le modèle (intérêt uniquement informatique).

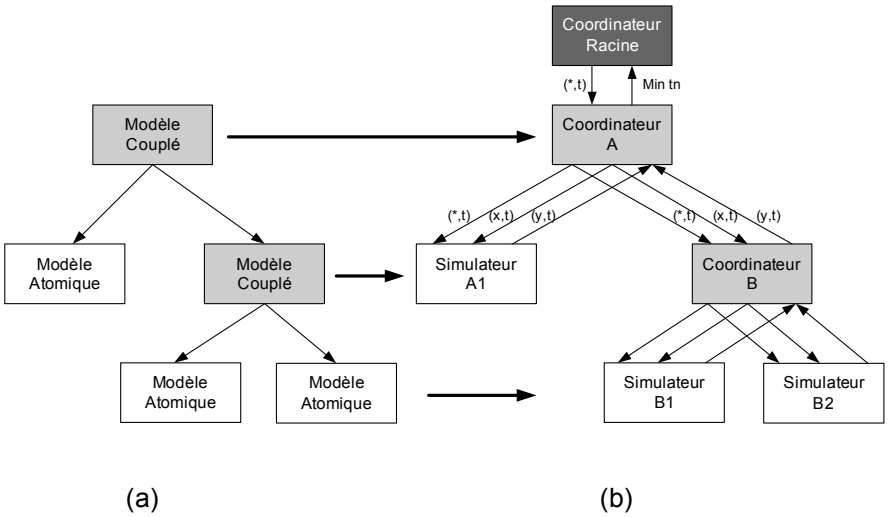


Figure 5 - Correspondance entre modèle et simulateur

Nous reviendrons en détail sur la simulation des modèles DEVS au cours du chapitre suivant.

2.4.2 Formalisme GDEVS

Les formalismes de spécification à événements discret traditionnels approximent les signaux d'entrée-sortie observés par une trajectoire constante par morceau (cf. Figure 6 haut). G-DEVS[4] définit les abstractions de signaux avec des trajectoires polynomiales par morceau [GIAMBIASI 95] (cf. Figure 6 bas). Ainsi, G-DEVS définit un événement comme une liste de valeurs. Ces valeurs représentent les coefficients du polynôme qui approchent les trajectoires d'entrée-sortie. Un modèle DEVS est donc un cas particulier du formalisme G-DEVS, c'est à dire un modèle G-DEVS d'ordre zéro (les trajectoires d'entrée-sortie sont constantes par morceau).

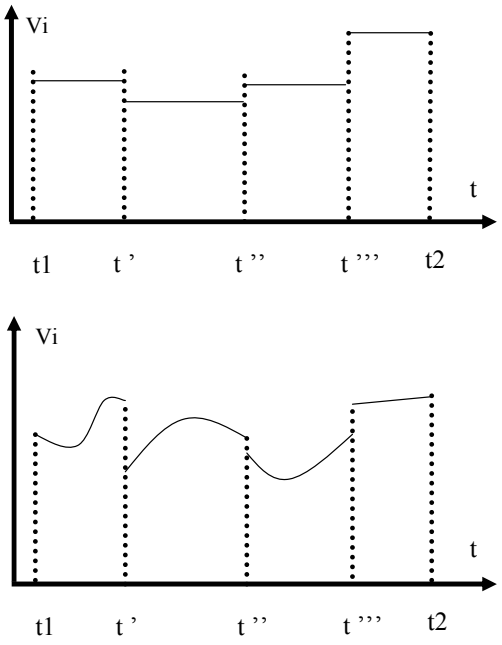


Figure 6 - Exemples de trajectoires constantes et polynomiales par morceau

[4] Generalized Discrete Event Specification

Formellement, un système dynamique représenté par un modèle à événements discrets G-DEVS d'ordre^N est défini comme suit :

$$DES_N = <XM, YM, S, \delta_{int}, \delta_{ext}, \lambda, D, Coef.>$$

Les descriptions suivantes sont nécessaires :

XM = A^{n+1}, où A est un sous-ensemble de nombres entiers ou réels

YM = A^{n+1}

S = $Q \times (A^{n+1})$

Pour tout état total (q, (a_n, a_{n-1},....., a_0), e) (avec e: temps écoulé dans S, $0 \leq e \leq D(S)$) et un segment polynomial continu w : $<t_1, t_2> \rightarrow X$, sont définis :

La fonction de transition interne :

$\delta_{int}(S) = \delta_{int}$ (q, (a_n, a_{n-1},....., a_0)) = $Straj_{q, x}$ (t1+D((q, (a_n, a_{n-1},....., a_0)), x))

avec $x = a_n t^n + a_{n-1} t^{n-1} + \ldots\ldots + a_1 t + a_0$

et Straj est la trajectoire de l'état du modèle

$\forall$ q of Q et $\forall$ w :$<t_1, t_2> \rightarrow X$,

$Straj_{q,w}$:$<t_1, t_2> \rightarrow Q$

La fonction de transition externe :

$\delta_{ext}(S, e, XM) = \delta_{ext}$ (q, (a_n, a_{n-1},.., a_0), e, (a_n', a_{n-1}',.., a'_0)) = ($Straj_{q, x}$ (t_1+e), x')

avec: Coef (x) = (a_n, a_{n-1},....., a_0)

et Coef (x') = (a_n', a_{n-1}',....., a'_0)

Coef : fonction qui associe les n-coefficients de la fonction polynomiale continue du segment w sur un intervalle de temps $<t_i, t_j>$, à (n+1) valeurs constantes (a_n, a_{n-1},....., a_0) de la façon suivante :

$w(t) = a_n t^n + a_{n-1} t^{n-1} + \ldots + a_1 t + a_0$

InCoef: fonction inverse :

InCoef (a_n, a_{n-1},..., a_0) = $a_n t^n + a_{n-1} t^{n-1} + \ldots + a_1 t + a_0$

La fonction de sortie :

$\lambda(S) = \lambda$ (q, (a_n, a_{n-1},.., a_0)) = (a'_n, a'_{n-1},.., a'_0)

La fonction définissant la durée de vie des états :

D(S) = **D** (q, (a_n, a_{n-1},..., a_0)) = MIN(e/Coef ($Otraj_{q, x}$ (t_1)) $\neq$ Coef ($Otraj_{q, x}$ (t_1 + e))

avec Otraj la trajectoire de sortie du modèle :

$Otraj_{q,w}$:$<t_1, t_2> \rightarrow Y$

2.4.3 Classe de modèles G-DEVS$_{PF}$

Une partie du travail abordé dans cette thèse se base sur la représentation en graphe de modèles atomiques G-DEVS. Nous allons donc introduire une classe de modèles G-DEVS qui peut être représentée par un graphe. Cette classe doit permettre de définir un modèle composé d'ensembles finis de nœuds et d'arcs. L'ensemble fini des nœuds est décrit par la variable d'état PHASE, qui est défini sur un sous ensemble fini de valeurs (nommées chacune explicitement) de l'ensemble des états. Il est ainsi possible de donner une sémantique plus directe à ces sous-ensembles. Elle permet également de représenter par un graphe les modèles G-DEVS par un modèle phase – transition (représentant les arcs entre les nœuds).

Nous nommons cette classe G-DEVS$_{PF}$ (modèles G-DEVS à Phase Finie) dont nous donnons ci dessous une description formelle.

$$\text{G-DEVS}_{PF} = < X, S_{PF}, Y, \delta_{int}, \delta_{ext}, \lambda, D >$$

X : ensemble des types des événements externes,

$S_{PF} = Q_{PF} \times (A^{n+1})$, avec :

$Q_{PF} = PHASE \times V$ où

PHASE = $\{Ph_1, \ldots, Ph_i, \ldots, Ph_n\} \neq \emptyset$ / $i \in \{\mathbb{N}\}$, est un ensemble fini non vide de valeurs symboliques.

$V = (\{v_{11}, \ldots, v_{1i}, \ldots, v_{1n}\ldots\} \times \{v_{21}, \ldots, v_{2i}, \ldots, v_{2n}\ldots\} \ldots \{v_{j1}, \ldots, v_{ji}, \ldots, v_{jn}\ldots\})$ / $i, j \in \{0,\ldots, n\}$, est un ensemble d'ensembles de variables d'état quelconques.

$A^{n+1} = (a_0 \times \ldots \times a_{n+1})$ (avec $a_i \in \mathbb{N}$ / $i \in \{0,\ldots, n+1\}$), représentent le (n+1)-uplet des coefficients du polynôme du dernier événement externe arrivé.

Y : ensemble des types d'événements de sortie

δ_{int} : $S_{PF} \rightarrow S_{PF}$,

δ_{ext} : $\mathbf{ST} \times \mathbf{X} \rightarrow S_{PF}$, avec : $\mathbf{ST} = \{(s_{pf}, e)\ s_{pf} \in S_{PF}, 0 \leq e \leq D(s)\}$,

λ : $S_{PF} \rightarrow \mathbf{Y}$ fonction de sortie,

D : $S_{PF} \rightarrow \mathbb{R}+ \cup \infty$ fonction durée de vie des états.

Il est à noter que cette classe permet également de représenter graphiquement des modèles DEVS, dans ce cas $A = a_0$.

2.4.3.1 Représentation graphique associée à un modèle atomique G-DEVS$_{PF}$

La représentation graphique, sous la forme d'un graphe de phases, permet d'exprimer aisément la dynamique d'un modèle atomique G-DEVS$_{PF}$. Les modélisateurs qui souhaitent définir des modèles DEVS ou G-DEVS avec un niveau d'abstraction élevé trouvent un intérêt dans la définition de tels modèles « graphiques ».

Une rationalisation de cette représentation a été proposée par [SONG 94] pour les modèles DEVS. Elle définit des symboles graphiques qui correspondent aux phases du modèle et aux transitions. Dans cette notation, les phases du modèle sont représentées par les nœuds du

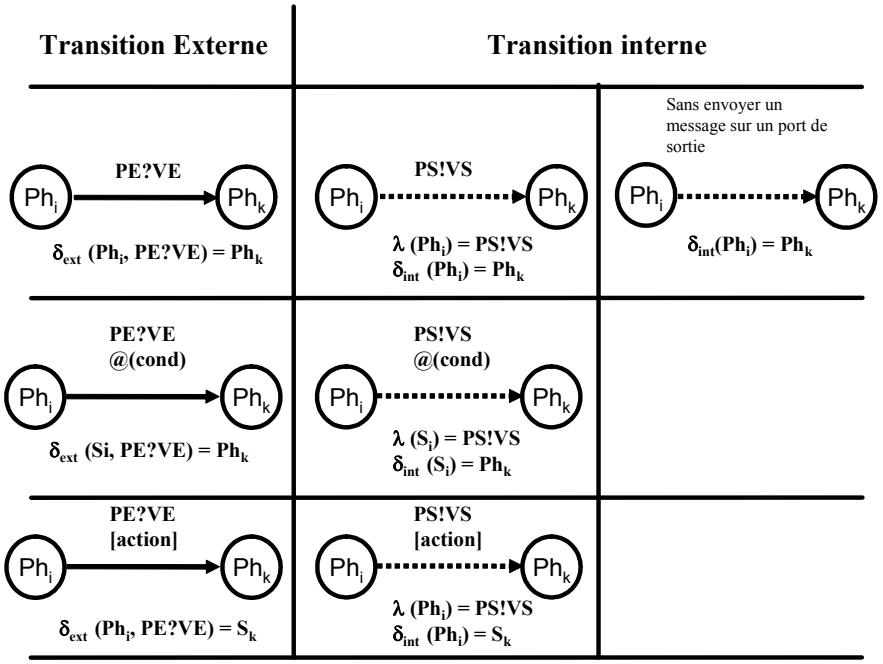
graphe, où l'on note également la durée de vie de la phase (D (Ph) = ddv). Les arcs représentent les transitions. Il existe donc deux types d'arcs :

Transitions externes : elles sont représentées par des arcs en trait plein. Sur l'arc, sont indiqués le port d'entrée ainsi que la valeur de l'événement d'entrée qui est attendue pour franchir cette transition. Toutes les transitions externes possèdent la structure suivante :

Valeur d'Entrée ? Port d'Entrée

Transitions internes : elles sont représentées par des arcs en trait pointillé. Sur un arc, l'étiquette mentionne la (ou les) sortie(s) générée(s) par le modèle, ainsi que le port de sortie associé : **Port de Sortie ! Valeur de Sortie**

La première ligne de la Figure 7 présente une synthèse de cette représentation.



Avec : VE = Valeur d'Entrée, PE = Port d'Entrée, VS = Valeur de Sortie, OP = Port de Sortie
cond = condition sur les variables d'état, action = modification des valeurs des variables d'état,
Ph = phase, V = autres variables d'état, A^{n+1} = coeff. dernier événement, S = (Ph, V, A^{n+1})

Figure 7 - Représentation Graphique d'un Modèle DEVS Atomique [SONG 94]

Il est important de garder à l'esprit que cette représentation est évidemment limitée aux modèles à phase explicite, c'est-à-dire aux modèles possédant une variable d'état nommée phase définie sur un ensemble fini de valeurs symboliques.

Il peut s'adjoindre, à la définition de l'état des modèles, d'autres variables d'états (appartenant à l'ensemble V et A^{n+1} définit précédemment) pour définir un G-DEVS dont l'ensemble de définition peut être infini. Dans ce cas, la notation graphique ne représentant initialement qu'une seule variable d'état, les arcs de transitions devront comporter des conditions portant sur la (ou les) variable(s) d'état supplémentaires pour permettre de la (les) prendre en compte dans les fonctions de transitions (deuxième ligne Figure 7). Lors d'une transition les valeurs des variables d'états de l'ensemble V, peuvent également être modifiées, pour cela on définit sur le modèle graphique des actions permettant de modifier les valeurs de ces variables d'état (deuxième ligne Figure 7). Notons que, dans le cas de l'utilisation de plusieurs variables d'état, on pourra définir la durée de vie de la phase par **D** (Ph, V, A^{n+1}) = ddv.

2.4.3.2 Notation graphique des modèle couplés G-DEVS

Les modèles atomiques ou couplés définis avec les formalismes DEVS / G-DEVS sont modulaires ; toute interaction avec l'environnement s'effectue par l'intermédiaire de ports d'entrée-sortie. Ce concept de port est fondamental car il permet, entre autres, de stocker des modèles en bibliothèque et de les réutiliser dans des descriptions hiérarchisées sans nécessairement connaître leur représentation interne [GIAMBIASI 95].

Ce concept a été formalisé par [CARSTEN 94] qui donne une représentation graphique de modèles couplés DEVS (Figure 8). Dans cette représentation, les modèles contenus dans le modèle couplé considéré sont représenté en bleu, les ports d'entrée et de sortie sont définis par les flèches à gauche et à droite des modèles, enfin les relations de couplage sont les lignes continues reliant des ports. Cette représentation a également été retenue pour la représentation des modèles couplés G-DEVS [ESCUDE 00].

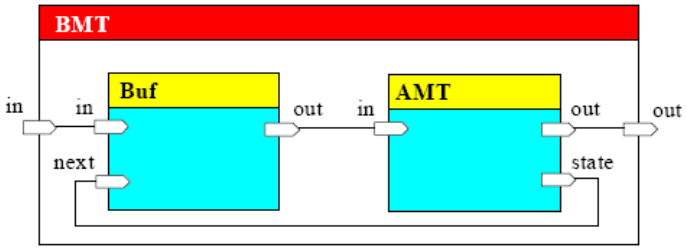


Figure 8 - Représentation Graphique d'un Modèle DEVS couplé [CARSTEN 94]

2.5 Conclusion Modélisation & Simulation

Après avoir rappelé les principales notions sur la modélisation et la simulation, nous avons constaté que les méthodes généralement préconisées pour modéliser et simuler les systèmes complexes se décomposent en plusieurs classes se différenciant sur les manipulations des variables d'état et de la représentation du temps.

Nous avons, ensuite, présenté les formalismes DEVS et G-DEVS qui sont à la base des travaux que nous présentons par la suite. Ces formalismes présentent l'intérêt d'une sémantique opérationnelle claire et précise et permettent la spécification de modèles de simulation sans aucun détail d'implémentation logicielle. Il est important de préciser qu'il s'agit bien de formalismes de spécification et non pas de langages de programmation.

3 Simulation distribuée

La simulation distribuée est apparue pour répondre à plusieurs attentes. Elle permet d'utiliser au mieux la puissance des ordinateurs, de travailler sur des ordinateurs distants et de réutiliser des simulations existantes en les interconnectant. Il existe plusieurs catégories de simulations distribuées que nous présentons ci-dessous.

3.1 Solutions architecturales disponibles

Parmi les solutions architecturales disponibles, nous pouvons retenir l'utilisation d'un ordinateur multiprocesseur qui partage sa mémoire. Il est également possible d'interconnecter

plusieurs ordinateurs éventuellement distants (*clustering*). Nous opterons pour cette possibilité qui offre le choix d'utiliser ou non une mémoire partagée, et fonctionne principalement par envoi de messages permettant aux différentes simulations d'échanger des données et de se synchroniser.

3.2 Méthodes de distribution possibles

Différents éléments peuvent être distribués dans une simulation. La première solution est la distribution des fonctions du simulateur. Une autre solution consiste à distribuer les événements. Enfin, la dernière solution consiste à distribuer les éléments du modèle. Dans ce cas, on procèdera par échange de messages datés entre les processus, il sera donc obligatoire d'utiliser un mécanisme de synchronisation des messages échangés entre les processus. On utilise, dans ce dernier cas, au mieux le parallélisme du modèle [FUJIMOTO 00].

3.3 Simulation distribuée a événements discrets

3.3.1 Evolutions

De façon similaire à la simulation séquentielle, l'évolution des simulations parallèles (ou distribuées) à événements discrets peut être synchrones ou asynchrones. Pour l'évolution synchrone, le temps simulé correspond à une horloge globale. Toutes les horloges logiques locales (horloges de chaque processeur) sont donc dirigées par cette horloge globale. A chaque pas temporel, les différents processeurs logiques (LP[5]) exécutent des actions.
Dans le cas d'une évolution asynchrone, le temps logique de chaque LP évolue d'événement en événement. Mais les LPs ayant entre eux une relation « d'influencés-influenceurs » doivent, avant toute exécution d'un événement daté au temps T, être sûrs de ne pas recevoir dans le futur un message avec une date T'< T. Cette contrainte est la causalité.

3.3.2 Exécution distribuée : respect de la causalité

Un traitement en simulation distribuée doit s'assurer de reproduire les relations de causalité temporelles existantes dans l'exécution séquentielle équivalente.

La causalité temporelle impose donc un ordre partiel entre les événements. [LAMPORT 78] définit une méthode basée sur les horloges logiques locales de telle façon que, pour tout événement a et b, si a est arrivé avant b (a → b), implique que le temps logique local C(a) doit être strictement inférieur à C(b).

3.3.3 Types de synchronisation

Pour respecter la causalité, deux types de synchronisations entre les LP sont possibles. Ces simulations supposent que l'on considère la simulation comme composée d'un ensemble de LPs qui communiquent en échangeant des messages datés (événements). Chaque LP exécute une simulation à événements discrets locale, qui maintient un échéancier avec les évé-

[5] Logical Processor

nements locaux et ceux reçus des influenceurs, un état local courant et une horloge locale qui contient la date logique locale actuelle.

3.3.3.1 Approche pessimiste (ou conservative)

Un LP sélectionne l'événement local (ou celui reçu d'un LP qui l'influence) dont la date T est la date minimum de son échéancier. Il traite cet événement en planifiant un nouvel événement local ou à destination de ses influencés, en modifiant la valeur de zéro ou plusieurs de ses variables d'état, et en actualisant sa date logique actuelle avec la date de l'événement traité.

Lorsqu'un LP traite un événement de date T, il doit être sûr de ne pas recevoir d'autres événements de date T' < T, et donc respecter le principe de causalité. L'objectif de ces premiers algorithmes de synchronisation est donc de déterminer les événements « sûrs » à traiter. Les premiers algorithmes permettant d'assurer lors du traitement d'un événement que le LP ne recevra pas d'autres événements de date inférieure ont été proposés par [BRYANT 77] et [CHANDY 79].

Dans cette solution, les LPs sont reliés par des liens (ou canaux) qui définissent la structure des relations d'influence entre LPs. Sur la Figure 9, les cercles représentent les LPs et les flèches les liens. Il est supposé que les LPs émettent des messages (des événements) sortant dans un ordre non décroissant et que le réseau assure que les messages soient reçus dans le même ordre que lors de l'émission. Cette supposition garantie que le dernier message reçu sur un lien est une date minimum par rapport à la date des prochains messages à recevoir sur ce lien. Enfin le LP sélectionne le message de date minimum parmi ceux reçus sur ces liens influenceurs et ses messages locaux.

Cependant, un problème survient dans les synchronisations conservatives si un des liens est vide lors de la détermination du prochain message à traiter. Dans ce cas, on ne connaît pas la date du prochain message à recevoir par ce lien et le LP attend indéfiniment cette information, cela se traduit par un interblocage (*DeadLock*), la Figure 9 illustre deux situations d'interblocage.

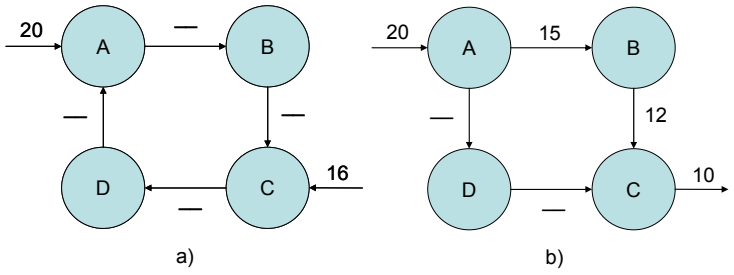


Figure 9 - Exemple d'interblocage [SAMADI 84]

Pour supprimer ces situations d'interblocage, [CHANDY 79] et parallèlement [BRYANT 77] ont introduit la notion de *Null message*. Un Null message n'a pas de contenu autre qu'une date. Lorsqu'un processeur transmet un message daté sur une de ses sorties, il transmet également un Null message de même date sur ses autres sorties. Dans la Figure 9 b), le LP C possède un canal vide, A envoie un Null message de date 15 vers D lors de l'émission du mes-

sage a destination de B ; a son tour, D émet un Null message de date au moins égale à 15 pour mettre à jour le contenu du lien entre D et C. Enfin C peut traiter sûrement son message de date 12 car il ne sera pas influencé par ailleurs avant 15.

Il est à noter que cette solution n'est pas suffisante dans le cas de LPs s'influençant en boucle, comme illustré Figure 9 a). Dans ce cas, l'un des LPs devra également avoir un temps simulé de traitement des messages non nul (de valeur ε). C'est à dire un délai non nul entre la réception d'un message et l'émission d'un message résultant de ce premier. Dans l'exemple a), le LP C pourra simuler le message de date 16 si $16 < 15+2\varepsilon$.

La solution utilisant les Null message est coûteuse en terme d'échange de messages, une solution alternative [CHANDY 81], consiste à attendre l'occurrence d'une situation d'interblocage puis, en identifiant le lien responsable de ce blocage, à proposer une solution de redémarrage.

Une dernière solution consiste à définir des barrières logiques ou des fenêtres temporelles entre les LPs.

Les contraintes de l'approche conservative proviennent en particulier de la non–exploitation du parallélisme total de la simulation, en effet, si un événement Ea sur un processeur A est susceptible d'affecter directement ou indirectement un événement Eb sur un processeur B, l'approche conservative doit forcement exécuter Ea et Eb séquentiellement. Si en pratique, Ea n'affecte pas Eb, ils auraient pu être traités de façon concurrente. Souvent le pire scénario est choisit pour déterminer s'il est « sûr » de traiter un événement, ce qui est de loin plus pessimiste que dans la pratique et force la séquentialisation même lorsque cela n'est pas nécessaire.

Pour résumer, la performance est fortement liée au choix d'une durée pendant laquelle les LPs attestent qu'ils n'émettront pas de message de sortie (ε dans la Figure 9 a), libérant ainsi de leur contrainte les LPs qu'ils influencent pour cette durée. Cette durée, appelée *Lookahead*, est dépendante du LP et du modèle à simuler.

3.3.3.2 Approche optimiste

En contraste avec l'approche précédente, l'approche optimiste permet la violation de la contrainte de causalité. Le LP traite les événements « à sa connaissance », cette connaissance locale et donc partielle des événements à traiter peut induire l'omission de certains événements externes et le non respect de la causalité [SAMADI 84]. Si une contrainte de causalité est enfreinte (réception d'un événement dont la date est antérieure à la date actuelle locale du LP), la simulation « revient alors dans son passé ». Un mécanisme de *Rollback* est alors déclenché [JEFFERSON 85].

On peut noter qu'à tout moment le message non traité contenant la date minimale parmi tous les échéanciers de tous les LPs est « sûr ». A partir de ce postulat, [JEFFERSON 85], dans *Time Warp*, définit donc la date de ce message non traité comme étant le temps minimal

global (*GVT*[6]). GVT est donc la date minimale jusqu'à laquelle la simulation pourra revenir. Toutes les données (état atteint, événements mémorisés) concernant une date inférieure sont supprimés. Le Temps Virtuel Global permet la libération de la mémoire occupée par les sauvegardes, la validation définitive d'actions, et ainsi l'exploitation des résultats de simulation jusqu'au GVT.

Les lacunes de cette approche optimiste proviennent notamment du temps d'exécution passé à faire des *Rollbacks* souvent trop important, et la nécessité de mémorisation des événements reçus, émis et des états atteints.

Il est à noter qu'il existe une troisième approche combinant les deux premières en fonction du coût de l'utilisation d'une méthode ou d'une autre sur la simulation.

3.4 Architectures de simulation

De l'utilisation grandissante des simulations distribuées, des environnements ou des normes visant à définir et soutenir leurs constructions sont apparus.

3.4.1 DIS

L'objectif de DIS[7] [DIS 94] est de relier des simulations hétérogènes. DIS modélise le monde comme un ensemble d'entités qui interagissent les unes avec les autres au travers d'événements. Ces événements sont créés par certaines entités et peuvent être perçus par les autres. A la base de DIS, on trouve un ensemble de protocoles qui permettent d'envoyer des messages au sujet des entités et des événements sur un réseau reliant entre eux les « nœuds » de simulation, c'est à dire les plates-formes informatiques ou réelles, responsables du maintient des entités des « Mondes » virtuels. Ces systèmes peuvent être répartis géographiquement.

Concrètement, DIS n'est pas uniquement un outil d'interconnexion de plates-formes de simulation ; il possède également une influence non négligeable sur les applications de ses simulations (à travers la définition des entités par exemple).

DIS était originellement conçu pour simuler des missions militaires (jeux de guerre, entraînement de troupes…), son utilisation s'est ensuite orientée vers le domaine civil avec des applications pour l'aviation civile, l'organisation des secours en cas de sinistre, etc.

Enfin, DIS a été défini initialement pour faire interagir des simulations dont la base de temps est continue, calée sur le temps réel avec une supervision humaine.

3.4.2 ALSP

Le protocole de Simulation de Niveau Global, ALSP[8] [MILLER 95], est à la fois un logiciel et un protocole, il est utilisé pour permettre à des simulations disparates de communiquer

[6] Global Virtual Time

[7] Distributed Interactive Simulation

[8] Aggregate Level Simulation Protocol

entre elles. Il est utilisé largement par l'armée américaine pour relier des simulations analytiques et d'entraînement pour répondre aux exigences d'entraînement des Corps de l'armée.

ALSP consiste en trois composants :

- le Logiciel d'Infrastructure ALSP (AIS) fournissant le support et la gestion d'un environnement d'exécution de simulation distribué ;
- une Interface ALSP réutilisable composée d'un jeu de protocoles de messages d'échange de données génériques (c'est-à-dire, des règles formelles pour l'échange d'informations) pour permettre l'interaction parmi les objets représentés dans des simulations différentes;
- des simulations participantes adaptées à l'utilisation avec ALSP.

3.4.3 HLA

3.4.3.1 Définition et but de HLA

L'Architecture de Haut Niveau HLA[9] est une spécification d'architecture logicielle permettant de créer des simulations comprenant différents composants de simulation [DMSO 98a]. Elle a été crée par le DMSO[10] du département DoD[11] pour des besoins de projet militaires.

Beaucoup de simulations complexes impliquent la combinaison de simulations individuelles issues de plusieurs types de systèmes, avec de plus, des aspects de l'environnement global à simuler (comme une représentation de joueurs humains interagissant dans un milieu avec des éléments physiques, etc.). L'illustration de ces problèmes est donnée par [CALVIN 96] et [WEATHERLY 98]. Fréquemment les simulations de certains de ces composants existent déjà, ayant été développées dans un but particulier ; il serait donc intéressant de les réemployer dans une nouvelle simulation globale. Malheureusement, il est souvent nécessaire de faire de vastes modifications pour adapter le modèle de simulation à base de composants pour qu'il puisse être intégré dans une nouvelle simulation composite. Dans certains cas, on peut même trouver plus aisé de mettre en œuvre une nouvelle simulation d'un composant du système que de modifier l'existant. Autrement dit, les modèles de simulation traditionnels manquent souvent de deux propriétés désirables : la réutilisation et l'inter fonctionnement.

La réutilisation, comme le nom le suggère, signifie que les modèles de composants de simulation peuvent être réutilisés dans des scénarii de simulation et des applications différents. Les composants de simulation réutilisables peuvent être combinés avec d'autres composants sans besoin de re-codage.

L'inter fonctionnement implique la capacité de combiner des simulations composantes sur différents types de plates-formes de calcul distribuées, souvent avec des opérations en temps réel. Cette approche implique de repenser la façon dont les composants de simulation

[9] High Level Architecture

[10] Defense Modelling an Simulation Office

[11] Department of Defense

agissent réciproquement contrairement à un programme monolithique dans un environnement informatique centralisé. Plutôt qu'une exécution de programme sur un ordinateur, il faut penser à un certain nombre d'exécutions de programmes sur différents types d'ordinateurs distribués avec des interactions au travers d'un système distribué en temps réel d'exploitation.

Dans HLA, chaque simulation participante est appelée **fédéré** ; elle interagit avec d'autres simulations fédérées au sein de ce qui est nommé dans HLA une **fédération**, qui est en fait un groupe de fédérés. Il est à noter qu'un fédéré n'est pas une simulation au sens le plus strict du terme. En effet, peuvent être considéré comme fédéré, des programmes observateurs ou faisant office d'interface avec le monde réel pour relever des informations ou interagir par exemple avec un opérateur humain.

Dans HLA, la communication est établie par partage, diffusion et réception d'informations. Ces informations sont représentées en termes de **Classes** issus de la programmation objet. Les classes aux données persistantes au cours de l'exécution sont nommées **Objet**. Ces objets possèdent des **Attributs** en termes de données membres. L'autre type d'information est nommé **Interaction,** de structure identique aux classes d'objets, son contenu est non persistant, on peut l'assimiler a la notion d'événement (simplement émis et reçu), ses champs sont nommés des **Paramètres**.

Cet ensemble de définitions a donné lieux à la création d'une norme HLA 1.3 en 1996, qui a ensuite évolué vers HLA 1516 en 2000. Des cours sont disponibles à [MCLEOD 99].

3.4.3.2 Les composants du standard HLA

HLA peut être défini selon trois composants [DMSO 98a] [IEEE1516 00] :

Les **Règles HLA** assurent l'interaction appropriée des simulations dans une fédération. Elles décrivent également les responsabilités des fédérations et des fédérés [IEEE1516 00].

Le **« Template » de Modèle d'Objet** (OMT[12]) fournit une structure commune pour la documentation de modèle d'objet HLA et favorise l'inter-fonctionnement et la réutilisation de simulations et de leurs composants [IEEE1516.1, 00].

La **spécification d'interface** définit les interfaces fonctionnelles entre les fédérés et l'infrastructure de temps d'exécution (RTI[13]) qui devront être respectées pendant l'exécution pour obtenir une simulation compatible HLA [IEEE1516.2, 00]. Le RTI est l'implémentation de cette spécification. Il fournit les services logiciels nécessaires pour une simulation compatible HLA (*HLA-compliant*).

Les règles de HLA

Au plus haut niveau, [IEEE1516 00] définit HLA en un jeu de dix règles auxquelles doivent obéir le fédéré ou la fédération s'ils souhaitent être considérés comme compatibles HLA. Les règles de HLA sont divisées en deux groupes consistant en cinq règles pour les fédérations HLA et cinq règles pour les fédérés HLA.

---

[12] Object Model Template

[13] Run Time Infrastructure

Règles des fédérations

Les fédérations auront un FOM[14], documenté conformément à l'OMT. Toute la représentation d'objets dans le FOM sera contenue dans le fédéré, non dans le RTI. Pendant une exécution de fédération, tout échange de données FOM parmi les fédérés se fera via le RTI. Pendant une exécution de fédération, les fédérés agiront réciproquement avec le RTI conformément à la spécification d'interface HLA. Pendant une exécution de fédération, un attribut d'une instance d'un objet appartiendra, à une date donnée, seulement à un fédéré.

Règles des fédérés

Les fédérés auront un SOM, documenté conformément à l'OMT.
Les fédérés seront capables de mettre à jour et/ou refléter n'importe quels attributs d'objets dans leur SOM et enverront et/ou recevront des interactions SOM extérieurement, comme spécifié dans leur SOM.
Les fédérés seront capables de transférer et/ou d'accepter la propriété d'attributs dynamiquement pendant une exécution de fédération, comme spécifiée dans leur SOM.
Les fédérés seront capables de varier les conditions dans lesquelles ils fournissent les mises à jour des attributs d'objets, comme spécifié dans leur SOM.
Les fédérés seront capables de gérer le temps local de façon à permettre de coordonner l'échange de données avec d'autres membres d'une fédération.

Les règles de fédération établissent les règles du jeu pour créer une fédération, y compris les exigences de documentation (Règle 1), la représentation d'objet (Règle 2), l'échange de données (Règle 3), l'intervention face aux exigences (Règle 4) et l'attribution de propriété (Règle 5). Les règles des fédérés traitent avec les fédérés individuels. Elles couvrent la documentation (Règle 6), le contrôle et le transfert d'attributs d'objet appropriés (Règles 7, 8 et 9) et la gestion du temps (Règle 10).

Le patron de Modèle d'Objet (OMT)

La réutilisation et l'inter-fonctionnement exigent que tous les objets et les interactions gérés par un fédéré, visibles à l'extérieur de ce fédéré, soient spécifiés en détail dans un format commun. Le « Template » de Modèle d'Objet (OMT) fournit une norme pour documenter l'information de Modèle d'Objet HLA. La spécification complète de cette norme peut être trouvée dans les documents [DMSO 98c] ou [IEEE1516.1 00] et [DMSO 98d]. Certaines tables sont définies de façon plus détaillée par [DMSO 98e]. La structure de l'OMT apparaît ci-dessous.

Modèle d'Objet de Fédération (FOM)

Il existe un seul FOM par fédération. Il présente toute l'information partagée. Les fédérés possède un certain nombre de données communes partagées au sein de la fédération. Ces données partagées sont recensées dans le FOM. Elles sont de deux types : les **objets** et les

[14] Federation Object Model

interactions. Les objets sont des informations partagées persistantes contrairement aux interactions qui sont des données éphémères (simplement émises et reçues). Enfin, ce modèle prépare les publications inter fédérées, par exemple, en prévoyant une entente du codage des données entrantes pour avoir un flux sortant interprétable par tous les fédérés intéressés par ces données. Les Objets et Interaction partagés dans une fédération sont définis sous forme de tables.

Modèle d'Objet de Simulation (SOM)

Chaque fédéré possède sont propre SOM[15], il décrit ici ses caractéristiques saillantes, c'est-à-dire ce qui sera partageable dans une fédération donnée. Il présente ainsi des objets et des interactions qui peuvent être employées extérieurement. Cette représentation se concentre sur les opérations internes que pourra fournir le fédéré. Les Objets et Interaction partageable par un fédéré sont définis sous forme de tables.

Modèle d'Objet de Gestion (MOM)

Le MOM[16] possède une Définition Universelle, il identifie les objets et les interactions qui gèrent une fédération.

Nous reviendrons plus en détail sur la construction des tables de SOM et FOM en présentant un exemple dans le chapitre 3. Les tables de l'application seront fournies en annexe.

La spécification d'interface

La spécification d'interface définit les interfaces fonctionnelles entre les fédérés et l'infrastructure de temps d'exécution (RTI). En effet, les informations échangées entre les fédérés dans une fédération ne sont pas échangés directement de fédéré à fédéré mais via un moteur de simulation HLA : le RTI (Règle 3).

Le RTI

Le RTI met en oeuvre la Spécification d'Interface HLA, c'est une base architecturale encourageant la portabilité et l'inter-fonctionnement. Le RTI est donc le logiciel qui se conforme à cette spécification, mais ce n'est pas une partie de cette spécification. Il fournit des services logiciels, qui sont nécessaires pour soutenir une simulation « HLA-compliant ».

La spécification d'interface identifie comment le fédéré agira réciproquement avec la fédération. A ce sujet, nous devons distinguer deux types de services dans l'interface :
Les services pour les fédérés (fournis par le RTI) sont les fonctions que le fédéré utilise pour interroger le RTI.
Les services fournis par les fédérés pour le RTI sont nommées fonction de « callback », ces fonctions sont implémentées dans le code du fédéré et doivent respecter, par héritage de fonc-

[15] Simulation Object Model

[16] Management Object Model

tions abstraites, une forme particulière pour être correctement appelées par le RTI, ces services sont représentés dans la littérature HLA par le symbole† (printer's dagger) succédant au nom du service.
La création d'un fédéré inclue donc la définition d'un certain nombre de fonctions pour l'usage du RTI. Une description des services RTI apparaît ci-dessous dans la version de la norme 1.3 définie dans [DMSO 98b] et 1516 dans [IEEE1516.1 00].
Enfin, les logiciels RTI sont nombreux et majoritairement commerciaux. Il existe cependant des RTI Open sources mais qui n'ont pas la certification des normes 1.3 ou 1516. Une liste des RTI existants peut être trouvée sur le site du DMSO [DMSO 05].

Les Services HLA

Ils séparent la simulation et la communication. Ce nouveau procédé améliore les standards plus vieux (par exemple, DIS, ALSP) [JENSE 97], il facilite la construction et la destruction de fédérations ; il soutient la déclaration d'objet et la gestion entre fédérés, aide la gestion du temps de la fédération. Enfin il fournit une communication efficace aux groupes de fédérés.
La spécification d'interface permet ainsi un ensemble de gestions :

- Gestion de la Fédération : permet à un fédéré de créer une fédération, de la rejoindre ou de la détruire. Permet également de gérer les mécanismes de sauvegarde restauration.
- Gestion des Déclarations : permet de définir les classes d'objet et d'interactions que les fédérés sont capables de mettre à jour ou pour lesquelles ils sont intéressés à souscrire. Ces concepts seront détaillés plus loin, mais à ce niveau il faut retenir que cette gestion concerne les informations échangées dans la simulation globale.
- Gestion des Objets : cette gestion permet la création, la mise à jour et la destruction des objets et des interactions. Les observations et modifications des objets et des interactions sont réalisées au travers du groupe précèdent.
- Gestion de la Propriété : cet ensemble de service permet de gérer la propriété des attributs des objets de la fédération. Il prévoit également les règles pour transférer ces propriétés d'un fédéré vers un autre.
- Gestion de la Distribution de Données : permet de définir de façons plus détaillée, au niveau des instances et des valeurs d'attributs les objets et interactions partagées.
- Gestion du Temps : permet de gérer l'évolution du temps (dirigé par le temps ou par les événements) dans la simulation et définit les types de synchronisation à utiliser (pessimiste optimiste), Les notions classiques de simulation distribuées comme le Lookahead…

3.4.3.3 Eléments de l'implémentation (architecture)

Un fédéré est un programme informatique compatible HLA, pour cela le code du fédéré conserve ses fonctionnalités originales mais il doit être complété par un certain nombre de fonctions pour permettre la communication avec les autres membres de la fédération. Ces

fonctions sont contenues dans le code de la classe *FederateAmbassador* qui permet de rendre interprétables par un processus local les informations reçues provenant de la fédération.

Le « Local RTI Components code » (LRC) fournit des fonctionnalités externes au fédéré pour l'enregistrement et l'utilisation des services du RTI tels que l'enregistrement d'objets et la gestion du temps, ces fonctionnalités sont fournies dans la classes *RTIAmbassador*.

Enfin, le composant central du RTI gère la fédération en s'appuyant notamment sur les informations fournies par le FOM pour définir les objets et les interactions participant à la fédération. La Figure 10 illustre une fédération composée de 2 fédérés A et B.

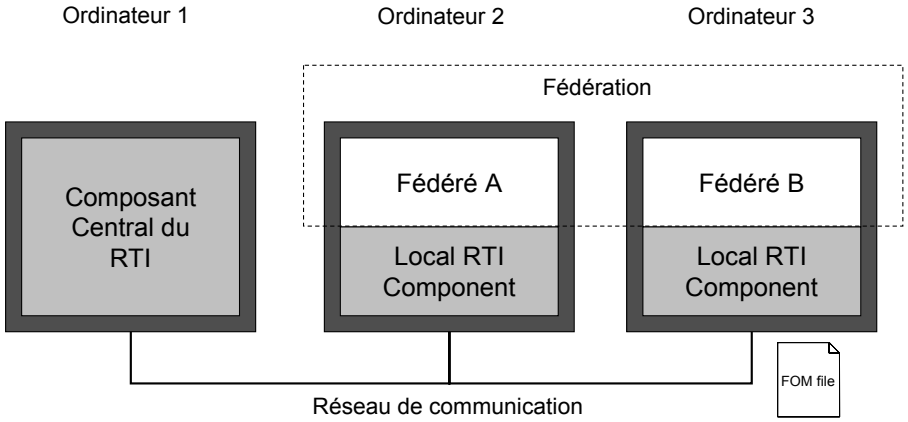


Figure 10 - Vue haut niveau, logique d'une exécution de Fédération HLA

3.4.3.4 Gestion de la Fédération

Il est ici question de création et de destruction de fédération, les services définit ici permettent aux fédérés de créer une fédération avec le service *createFederationExecution()*, si la fédération n'existe pas ou de la joindre avec *joinFederationExecution()*. Il est également possible pour un fédéré de quitter une fédération et de la détruire, à l'aide des mécanismes *ResignFederation()* et *DestroyFederation()*. La gestion des déclarations permet également de sauvegarder l'état d'une fédération et de revenir plus tard à ce point de sauvegarde. Ces derniers services ne sont pas détaillés ici.

3.4.3.5 Gestion des Déclarations

Un fédéré peut, au travers des services proposés par le RTI, publier *PublishObjectClass()* et souscrire *SubscribeObjectClass()* à une classe ou de la même façon à une interaction (*PublishInterractionClass()* et *SubscribeInteractionClass()*). Le terme Publier signifie que le fédéré a l'intention de diffuser la création d'instances de classe et la mise à jour des attributs de ces instances. Le terme souscrire signifie l'intention des fédérés de refléter certains attributs ou paramètres de certaines classes à d'autre fédérés.

3.4.3.6 Gestion des Objets

Ici sont définis les services permettant l'échange de données entre fédérés au travers du RTI. Ces messages représentent les événements de la simulation. Ces événements sont trans-

mis au RTI, ensuite le RTI gère la distribution des messages vers les fédérés intéressés par ces informations. Les services les plus utilisés sont *RegisterObject()* qui permet de créer un nouvel objet au sein de la fédération, Le callback associé est *DiscoverObjectInstance()†* qui permet d'informer les fédérés de cette création. Les autres services permettent de mettre a jour les valeurs des attributs d'un objet avec *UpdateAttributeValues()* et *ReflectAttributeValues()†* ainsi que l'émission réception d'interaction avec *SendInteraction()* et *ReceiveInteraction()†*. Il est également possible de s'informer de la modification des valeurs d'attributs d'objet à tout moment. Enfin cet ensemble de services permet également de gérer le type de transmission de messages, garantie ou non. Les transmissions garanties sont définies par l'appel aux services *changeAttributeTransportType(Reliable)* et *changeInteractionTransportType(Reliable)*, ce type de transport est assuré en utilisant le protocole TCP. Les transmissions non garanties de réception sont définies par l'appel aux services *changeAttributeTransportType(bestEffort)* et *changeInteractionTransportType(bestEffort)*, ce type de transport est assuré en utilisant le protocole UDP.

3.4.3.7 Gestion de la Propriété

Les fédérés créent des objets, ils ont, à ce moment, la propriété de leurs attributs. Les fédérés souhaitant acquérir la propriété d'attributs d'objets en font la demande auprès du RTI qui la transmet au fédéré concerné, un fédéré recevant une telle demande peut accepter de céder cette propriété ou non. Un fédéré peut également céder la propriété d'un attribut d'objet sans demande, L'attribut se retrouve dans ce cas sans propriétaire. Il est à noter que cette gestion d'objets n'est pas forcement utilisée dans toutes les fédérations.

3.4.3.8 Gestion de la Distribution de Données

Il est possible d'affiner l'échange des données entre fédérés en définissant des **régions** d'influence, en reprenant les mécanismes de publication et de souscription mais en travaillant en termes d'instances de classes d'objets ou d'interaction. On peut ici déterminer plus précisément les messages à échanger entre les fédérés et ainsi diminuer les échanges.

3.4.3.9 Gestion du temps

Il est possible, en se conformant à HLA, de choisir la façon dont vont être échangées les données au sein de la fédération. Les messages peuvent être de type **ReceiveOrder** (RO), dans ce cas les messages ne sont pas datés, ils sont simplement envoyés et reçus sans contraintes temporelles. Une autre possibilité consiste à définir des messages datés **TimeStampOrder** (TSO). Dans ce cas, HLA utilise les algorithmes de synchronisation pessimistes et/ou optimistes de la simulation distribuée vus précédemment. Le RTI implémente donc des notions associées à la synchronisation du temps dans ses deux versions, seuls les noms de certaines notions ou services diffèrent parfois entre la norme 1.3 [DMSO 98b] et 1516 [IEEE 1516.2 00]. Les fédérés émetteurs, c'est-à-dire qui publient, des messages TSO sont dits **régulateurs** de temps (*Time Regulating*) et les fédérés qui reçoivent, c'est-à-dire qui ont sous-

crit à, des messages TSO sont dits **contraints** de temps (*Time Constrained*). Il est à noter que les fédérés peuvent être à la fois régulateurs et contraints de temps.

HLA utilise la notion de **Lookahead.** Rappelons que le Lookahead L est la durée au delà de sa date actuelle pendant laquelle un fédéré atteste qu'il n'émettra pas de message. Les fédérés peuvent modifier la valeur de leur Lookahead en cours d'exécution. Dans le cas d'une augmentation par un fédéré de son L, le RTI prend en compte directement cette modification. Dans le cas d'une diminution le RTI prend en compte la demande mais elle ne sera effective qu'après que le fédéré est avancé son temps de l'ancienne valeur de son L, cela pour respecter la causalité.

A partir des Lookahead des fédérés, le RTI peut calculer pour chaque fédéré le **LBTS**[17] ou **GALT**[18]. C'est la date jusqu'à laquelle le fédéré ne sera pas influencé par ces influenceurs, c'est donc le minimum des dates actuelles de ses influenceurs plus leurs Lookahead.

Les méthodes d'avancement du temps dans HLA peuvent être divisées en deux catégories : par pas constant ou par événements.

La réponse du RTI dépend du mécanisme de synchronisation choisi pour faire évoluer le temps.

Dans le cas d'un avancement par pas de temps « time stepped federates », le pas d'avancement dans le temps est constant tout au long de l'exécution. Les fédérés effectuent la demande avec la fonction *TimeAdvanceRequest()* indiquant ainsi qu'ils se sont acquittés du traitement des messages de dates inférieures. En retour, le RTI accorde un *TimeAdvanceGrant()†* lorsque tous les fédérés ont effectuée leurs demandes d'accord d'avancement pour une même date.

Si l'avance est définie par événement « event driven federates », l'avance dans le temps du fédéré est conditionnée par la réception d'un message. Un fédéré demande à avancer au temps du prochain message TSO local ou reçu. Dans le cas d'une évolution dirigée par les événements, les fédérés interrogent le RTI pour obtenir l'autorisation de sélectionner un événement dans leurs échéanciers avec la fonction *NextEventRequest()* 1.3 ou *NextMessageRequest()* 1516. À cet appel le RTI répond en fonction du mécanisme de synchronisation choisi pour faire évoluer le temps. Dans le cas du choix d'une simulation pessimiste, le service *TimeAdvanceGrant()†* est activé uniquement si le RTI est sûr de transmettre les messages dans le bon ordre causal. Dans le cas où le RTI doit d'abord délivrer un message de date inférieure a la date d'avance demandée le RTI appellera *ReceiveInteraction()†* ou *ReflectAttributesValues()†* plus un *TimeAdvanceGrant()†*. Ces notions seront reprises et détaillées dans le chapitre 3.

La dernière possibilité est l'avancement optimiste, les fédérés avancent dans le temps à leur vitesse propre, indépendamment des autres. Il s'agit d'un mode d'avance dans le temps qui ne respecte pas la causalité et il constitue une exception aux principes de causalités. Dans

[17] Lower Bound on Time Stamp HLA version 1.3

[18] Greatest Available Logical Time HLA version 1516

ce mode, le RTI ne peut garantir que le fédéré ne recevra pas d'événements correspondant à son passé. Et quand une telle situation apparaît, c'est au fédéré de résoudre les incohérences en invoquant des services d'annulations de messages ou « Rollbacks », afin de revenir à un état de la simulation cohérent avec le traitement de l'événement.

Le **MNET**[19] ou **LITS**[20] d'un fédéré est une borne inférieure jusqu'à laquelle le fédéré ne recevra pas de message. Cette notion est similaire au LBTS, mais prend de plus en compte les messages dans la queue du RTI qui n'ont pas encore été délivrés.

Nous pouvons noter ici que la gestion du temps fournit des mécanismes de synchronisation qui restent à implémenter dans les fédérés afin de prendre en considération les messages, provenant de la fédération, en respect du temps simulé.

3.4.3.10 Le FEDEP

HLA propose un modèle de développement de fédérés et de fédérations basé sur les modèles en cycle du génie logiciel. Ce modèle est nommé FEDEP[21] [IEEE1516.3 03].

Dès l'origine de l'élaboration de la norme HLA, un besoin a été identifié dans la conception et le développement de fédérations HLA. Ce besoin est caractérisé par un haut degré de flexibilité dans le processus de composition et d'exécution d'applications HLA permettant de réaliser les objectifs d'une application « composée ». Le but de FEDEP est d'éviter la répétition de questions inutiles à chaque nouvelle construction de fédération sur la façon dont les applications HLA ont été construites et s'exécutent. En effet, il a été reconnu que le processus de développement et d'exécution des fédérations HLA peut varier significativement entre des applications crées par différents concepteurs.

Par exemple, le type et l'ordre des activités de bas niveaux exigés pour développer et exécuter des fédérations orientées analyse de données vont probablement complètement différer de celles exigées pour développer et exécuter un programme d'exercices d'entraînement distribués. Cependant, à un niveau plus abstrait, Le DMSO a identifié, dans la version 2003, un cycle de sept pas de base que toutes les fédérations HLA devraient suivre pour développer et exécuter leurs fédérations. La Figure 11 illustre chacun de ces pas et avec le récapitulatif suivant :

Pas 1 : Définition des objectifs de fédération : L'utilisateur de la fédération et l'équipe de développement de la fédération définissent et conviennent d'un ensemble d'objectifs et documentent ce qui doit être accompli pour réaliser ces objectifs.

Pas 2 : Réalisation d'une analyse conceptuelle : Basé sur les caractéristiques de l'espace d'un problème, une représentation appropriée du domaine du monde réel est crée.

---

[19] Minimum Next Event Time HLA version 1.3

[20] Least Incoming Time Stamp HLA version 1516

[21] Federation Development and Execution Process

Pas 3 : Conception de fédération : Les fédérés, réutilisables dans la fédération considérée, sont identifiés. Les activités pour la modification des fédérés et/ou la création de nouveaux fédérés sont exécutées. Les fonctionnalités exigées par la fédération sont allouées aux fédérés et un plan est défini pour le développement de la fédération et sa mise en œuvre.

Pas 4 : Développement de fédération : Le Modèle d'Objet de Fédération (FOM) est développé. Des accords entre les fédérés sont établis sur la base de données à partager et les algorithmes qui l'exploiteront. Enfin, les nouveaux fédérés et/ou les modifications des fédérés existants sont mis en œuvre.

Pas 5 : Planification, intégration et test de la fédération : Toutes les activités d'intégration nécessaires à la fédération sont exécutées et des tests sont réalisés pour assurer ces conditions d'inter-fonctionnement.

Pas 6 : Exécution de la fédération et préparation des résultats : La fédération est exécutée et les données produites par l'exécution de la fédération sont pré traitées. Ce pas est un rajout par rapport à l'ancienne norme, il marque la nécessité d'une phase de calibration de la classification des résultats.

Pas 7 : Analyse des données et évaluation des résultats : Les données produites par l'exécution de la fédération sont analysées et évaluées et les résultats sont présentés dans un rapport à l'utilisateur.

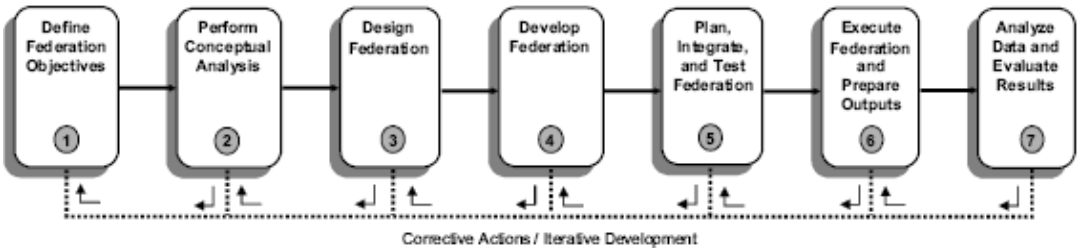


Figure 11 - Federation development and execution process (FEDEP), vue de haut niveau.

3.4.4 Techniques de simulation distribuée « non dédiées »

3.4.4.1 CORBA

L'architecture CORBA[22] est une architecture logicielle [VINOSKI 97], pour le développement de composants et d'Object Request Broker ou ORB. Les composants, qui sont assemblés afin de construire des applications complètes, peuvent être écrits dans des langages de programmation distincts, être exécutés par des processus séparés, voire être déployés sur des machines distinctes.

Les composants CORBA emploient une approche essentiellement orientée objet. Les applications et les composants CORBA mélangent le typage statique et dynamique. Ainsi, chaque composant est décrit statiquement par une interface mais les composants qui utilisent ce-

[22] Common Object Request Broker Architecture

lui-ci doivent vérifier dynamiquement que l'interface est effectivement implantée. CORBA est un standard maintenu par l'OMG[23].

3.4.4.2 Autres techniques de simulation distribuée « non dédiées »

Il existe d'autres types de simulations distribuées s'appuyant sur des techniques « non dédiées » à la simulation distribuée telles que les web services, le *peer to peer*, l'utilisation de *sockets*, etc. Cependant nous pensons que ces techniques de partage et de diffusion d'informations sont incomplètes car elles ne proposent pas de mécanismes de synchronisations nécessaires à l'échange de messages datés. Dans la perspective de leurs utilisations dans notre étude, il serait donc nécessaire de mettre en œuvre l'ensemble de ces mécanismes, remettant ainsi en cause l'expérience acquise et intégrée dans les normes dédiées à la simulation distribuée.

De part l'incapacité à manipuler les échanges d'informations datées, nous ne retiendrons pas les techniques non dédiées dans le choix d'une norme en soutient de la définition d'une simulation à événements discrets distribuée.

3.5 Conclusion Simulation Distribuée

HLA s'inscrit comme une norme définissant un environnement de modélisation et simulation. Cette norme se démarque des précédentes (DIS, ALSP…) et des techniques non dédiées par les propriétés de réutilisation et de gestion du temps développées dans cette norme. En effet, la réutilisation des objets partagés est facilitée par la définition de documents décrivant la structure de chaque objet partagé par un fédéré HLA. De plus, une partie des mécanismes de synchronisation d'échange d'information est déjà présente dans le logiciel (RTI) spécifié dans cette norme, l'intégration d'un programme dans un fédéré est donc facilitée par héritage de classes prédéfinies et appel de fonctions du RTI dédiées aux services de gestion du temps proposés par le RTI. Pour ces raisons, nous utiliserons la norme HLA en soutient du développement de l'environnement de simulation distribué introduit dans cette thèse.

Malgré cela, il est important de retenir que la norme HLA n'est pas une implémentation, ce n'est qu'une spécification dont on suit les instructions dans la conception et la réalisation de fédérés et de fédérations.

4 Workflow

4.1 Une Introduction au Workflow

Un *Workflow* est la modélisation et la gestion assistée par ordinateur de l'accomplissement des tâches composant un processus administratif ou industriel, en interaction avec divers acteurs (humains, logiciels, ou matériels) invoqués [COURTOIS 96]. Outil informatique d'origine industrielle, le Workflow est l'adaptation de la GED[24] adjoint de la

[23] Object Management Group

[24] Gestion Electronique des Documents

faculté à gérer l'échange de messages. Le Workflow propose des solutions d'optimisation et de rationalisation des flux d'informations ; que ces informations soient associées à des documents, des procédures ou des messages complémentant les systèmes de gestion électronique de documents et d'informations.

A l'heure actuelle plus de 250 Systèmes de Gestion de Workflow (WFMS) sont utilisés ou en développement. Cela signifie que le terme « gestion de Workflow » n'est pas simplement une nouvelle expression à la mode. Ce phénomène de gestion de processus (Workflow) aura certainement un fort impact sur la génération suivante de systèmes informatiques [COURTOIS 96, HAYES 91, KOULOPOULOS 95, SCHAEL 97].

4.2 Origines

Il est intéressant de considérer l'évolution des systèmes informatiques au cours des quatre dernières décennies [VAN DER AALST 02] pour prendre conscience de la pertinence d'une gestion électronique de processus (Workflow) et apprécier l'impact de la gestion de Workflow dans un avenir proche.

La Figure 12 présente le phénomène de gestion de Workflow dans une perspective historique. Cette figure décrit l'architecture d'un système informatique classique en termes de composants. Dans **les années soixante**, un système informatique était composé d'un certain nombre d'applications autonomes. Pour chacune de ces applications une interface utilisateur et un système de base de données spécifique étaient développés, chaque application possédait donc ses propres routines pour interagir avec l'utilisateur, stocker et récupérer les données. Dans **les années soixante-dix**, le développement des systèmes de gestion de base de données (SGBD) a permis d'extraire les données des applications. En utilisant les SGBD, les applications ont ainsi été libérées du fardeau de la gestion de données. Dans **les années quatre-vingts,** l'apparition de systèmes de gestion d'interface utilisateur « User Interface Management Systems » (UIMS) a permis aux développeurs d'application d'extraire l'interaction avec les utilisateurs des applications. Enfin, **les années quatre-vingt-dix** sont marquées par l'apparition de logiciels de Workflow, permettant aux développeurs d'application d'extraire les procédures de travail des applications. La Figure 12 fait apparaître le système de gestion de Workflow comme une composante générique pour représenter et manipuler les processus d'entreprise[25].

Ainsi, à l'heure actuelle, beaucoup d'organisations commencent à considérer l'utilité d'outils avancés pour soutenir la conception et l'exécution de leurs processus d'entreprise.

[25] Les procédures d'entreprise représentent l'organisation et la politique de l'entreprise pour atteindre certains objectifs [WFMC11 99]

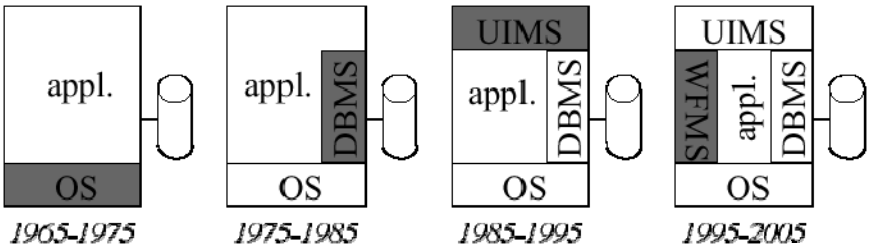


Figure 12 - Les systèmes de gestion de Workflow dans une perspective historique

4.3 Motivations

L'intérêt accru pour la gestion des processus d'entreprise s'explique par plusieurs faits. Tout d'abord, l'apparition de philosophies de gestion comme la Reconstruction de Processus d'Entreprise BPR[26] et l'Amélioration Continue des Processus a stimulé les organisations à prendre conscience des processus d'entreprise. Le deuxième élément motivant l'intérêt pour la gestion des processus provient du nouveau postulat : les organisations actuelles doivent être en mesure de proposer une large gamme de produits et de services indexées sur l'actualité des demandes. Ceci induit une augmentation du nombre de processus, de produits et de services à l'intérieur des organisations, et en contrepartie une diminution de la durée de vie des produits et des services. Tout cela entraîne en conséquence une demande de flexibilité et de réactivité accrue à l'entreprise qui ne peut être prise en compte qu'au travers d'une gestion rigoureuse des processus.

En conséquence, les processus d'entreprise actuels sont soumis à des changements fréquents. De plus, la complexité de ces processus a considérablement augmenté. Tous ces changements de l'environnement des organisations, ont fait des processus d'entreprise une préoccupation importante dans le développement de systèmes informatiques. Des solutions de gestion de processus existent dans ce domaine mais sont souvent ad hoc. Il est donc important de définir un modèle conceptuel et d'unifier les techniques de modélisation actuelle. En conclusion il existe aujourd'hui un réel besoin pour une nouvelle composante dans les entreprises ; cette composante se nomme « Système de gestion de Workflow ».

4.4 Définitions et terminologies

Les définitions sont, pour la majorité, issues de la Coalition de Gestion de Workflow « Workflow Management Coalition » (WfMC). La WfMC a été fondée en 1993 par un regroupement d'industriels de l'informatique, de chercheurs et d'utilisateurs, associée à l'essor du développement des Workflows. Cette coalition a pour but de promouvoir les Workflow et d'établir des standards pour les « Workflow Management System » (WfMS). Elle a en particulier publié un glossaire de référence contenant les terminologies employées dans ce domaine [WFMC03 95 ; WFMC11 99]. Ces standards servent notamment à résoudre les problèmes d'interopérabilité entre systèmes Workflow mais également à définir les caractéristi-

[26] Business Process Reengineering : Démarche de remise en question et de redéfinition en profondeur des processus d'une organisation en vue de la restructurer pour la rendre plus efficace tout en réduisant les coûts.

ques fondamentales de ces systèmes. Les documents publiés par la WfMC, qui couvrent plusieurs aspects, peuvent être considérés comme des références en la matière.

4.5 Définitions de base du Workflow

Le sens du mot Workflow peut varier en fonction du contexte. Pour plus de clarté, les définitions les plus communément admises sur les concepts et les termes du Workflow sont rappelées ci dessous. Ces définitions sont principalement issues du « Workflow Management Coalition Terminology and Glossary » WFMC-TC-1011 [WFMC11 99], dont il existe une traduction à usage francophone [WFMC03f 98].

L'idée première du Workflow est donc de séparer les processus, les ressources et les applications, afin de se recentrer sur la logistique des processus travail et non pas sur le contenu des tâches individuelles. Un Workflow est donc le lien entre ces trois domaines comme précise la Figure 13.

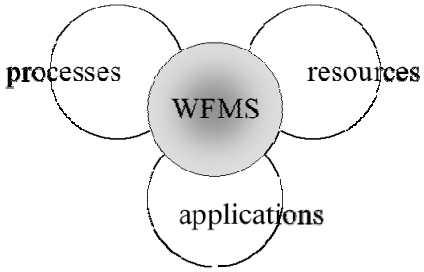


Figure 13 - Environnement du système Workflow

4.5.1 Définition d'un Workflow

Le Workflow est une technologie informatique ayant pour objectif la gestion des processus d'organisations ou d'entreprises : les termes suivants sont également employés pour qualifier cette technologie « Système de Gestion Electronique de Processus », « Gestion de Workflow » ou « Gestion de processus » [COURTOIS 96].

Le Workflow est l'ensemble des moyens mis en œuvre pour automatiser et gérer les processus d'une organisation. Cette gestion est rendue possible par la représentation sous forme d'un modèle, de tout ou partie des processus considérés. Le Workflow doit ensuite transcrire les modèles obtenus en une forme exécutable. Enfin, ces modèles sont exécutés et gérés. Il est ainsi possible de suivre l'évolution de leur état au fil du temps. La gestion de processus inclut également, au cours de l'exécution, la coordination et la synchronisation des différents acteurs des processus en fonction de l'état actuel des modèles.

Pour résumer, la Gestion de Processus permet donc d'attribuer à chacun et au bon moment, les tâches dont il a la responsabilité et de mettre à disposition les applications, les outils et les informations nécessaires pour leurs réalisations. Dans un contexte d'acteurs humains, le Workflow permet de décharger les acteurs de certaines tâches de gestion administrative, en leur laissant la possibilité de se concentrer sur les contenus des tâches techniques en rapport avec leurs compétences. De plus, le Workflow donne la possibilité d'effectuer une activité de

monitoring sur le déroulement des Workflow de l'entreprise, permettant en particulier de connaître, en fonction de la date, l'état des activités, des acteurs, des applications et quelles sont les prochaines activités planifiées.

En synthèse, La WfMC présente le Workflow comme l'automatisation d'un processus d'entreprise, en intégralité ou en partie, pendant laquelle on définit les transmissions des documents, de l'information ou des tâches d'un participant à un autre pour agir, selon un jeu de règles procédurales [WFMC11 99]. Un **Système Workflow** définit, gère et exécute des procédures en exécutant des programmes dont l'ordre d'exécution est prédéfini dans une représentation informatique de la logique de ces procédures - les Workflow [WFMC11 99].

4.5.2 Méta Modèle basique

Le Workflow est basé sur un ensemble de concepts. La WfMC [WFMC11 99] a proposé un méta modèle de Définition de Procédures, qui identifie les concepts de haut niveau dans la Définition de Processus. Ce modèle permet de mieux appréhender les concepts et leurs interrelations.

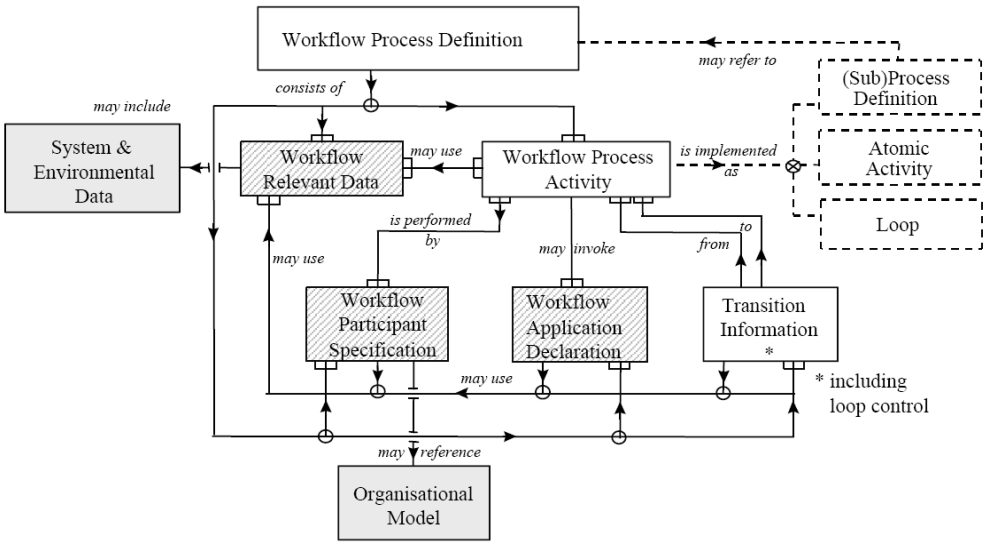


Figure 14 - Méta modèle Workflow pour la définition de Processus [WFMC11 99]

Le méta modèle, présenté Figure 14, identifie un ensemble d'objets fondamentaux qui entrent dans la définition d'un processus géré par un système Workflow, et que nous allons définir et commenter dans les paragraphes suivants. Remarquons que le méta modèle peut être enrichi par les développeurs de systèmes, il peut également être utilisé à des fins d'échanges entre différents systèmes Workflow.

4.5.3 Concepts et Terminologie Workflow fondamentaux

Les principaux termes associés aux Workflow proposés par la WfMC [WFMC11 99] sont présentés dans le diagramme du méta modèle Workflow ci-dessus, ce diagramme permet également de mettre en évidence leurs interrelations. Les termes présentés ci-dessous en français

avec la traduction anglaise originale associée, couvrent les notions plus importantes appartenant au Workflow et à son lexique [WFMC11 99].

4.5.3.1 Procédure Workflow (*Workflow Process*)

Une procédure Workflow est une procédure contrôlée par un Workflow. Une procédure est composée de plusieurs activités enchaînées pour représenter un flux de travail. Une procédure possède une structure hiérarchique et modulaire, en l'occurrence une procédure peut donc être composée de sous procédures et d'activités. Les sous-procédures peuvent être composée elles mêmes de procédures manuelles ou de procédures Workflow.

4.5.3.2 Activité (*Process Activity*)

Une activité est une étape d'un processus au cours de laquelle une action élémentaire est exécutée. On désigne par « action élémentaire » (ou tâche) une activité qui n'est plus décomposable en sous-procédures. La WfMC distingue une « activité manuelle », qui n'est pas contrôlée par le système Workflow, et une « activité Workflow » qui est sous le contrôle du Workflow. Un exemple d'une activité manuelle est l'ouverture d'un courrier. Une activité Workflow peut être le remplissage d'un formulaire électronique. Il existe donc des exemples d'activités manuelles intégrables dans un Workflow.

[VAN DER AALST 98a] présente l'activité Workflow comme l'intersection entre une ressource humaine ou matérielle et un bon de travail dans le cadre de l'exécution d'une tâche. Dans cette représentation, une ressource du modèle organisationnel est donc exigée pour qu'une tâche puisse être instanciée en activité et allouée à un participant de Workflow.

4.5.3.3 Acteur, Ressource (*Workflow Participant*)

Un acteur est une entité du modèle organisationnel participant à l'accomplissement d'une procédure. L'acteur est chargé de réaliser les activités qui lui sont attribuées via le(s) rôle(s) qui lui sont définis dans le modèle organisationnel. Les autres dénominations courantes dans la littérature de cette entité sont « ressource », « agent », « participant » ou « utilisateur ». L'acteur peut être une ressource humaine ou matérielle (machine, périphérique informatique…).

Les ressources sont organisées en classes dans le modèle organisationnel. Ces classes sont des groupes de ressources possédant des propriétés communes. Une classe est basée sur :

Rôle : défini ci dans le § suivant.

Groupe : cette classification est basée sur l'organisation (département, équipe, unité).

4.5.3.4 Rôle (*Role*)

Un rôle décrit en général les compétences d'un acteur dans le processus ou sa position dans l'organisation. Un rôle est associé à la réalisation d'une ou de plusieurs activités. Plusieurs acteurs peuvent tenir un même rôle. La WfMC distingue deux types de rôles [WFMC11 99] :

Les rôles **organisationnels** définissent un ensemble de compétences qu'un acteur possède. Ce rôle définit la position de l'acteur dans une organisation.

Les rôles **procéduraux** définissent une liste d'activités qu'un acteur est en capacité d'exécuter.

Il est à noter que certains travaux ne différencient pas les notions d'acteur et de rôle et ne parlent que d'acteur. Cette opinion semble restreindre la clarté et la flexibilité des modèles Workflow.

4.5.3.5 Données (*Workflow Relevant Data*)

Une donnée pertinente pour les procédures est une information en rapport avec la réalisation des activités (en définition de la tâche, en entrée ou en sortie). Elle peut constituer l'objectif d'une tâche (manipulation de la donnée et définition de l'état de la procédure), être un élément essentiel pour activer les transitions d'état d'une instance Workflow ou être généré par la tâche et ainsi intervenir dans la détermination de la prochaine activité à déclencher. Ces données sont en général des objets au sens purement informatique mais peuvent également être une représentation d'objets physiques.

Notons qu'il existe deux autres types de données utilisées hors de la gestion de procédures :

Donnée de contrôle (Control Data) : données gérées et utilisées par le système Workflow et les moteurs Workflow.

Données Applicatives (Applicative Data) : données propres aux applications, le système de gestion de Workflow n'y a pas accès.

4.5.3.6 Application externe (*Invoked Application*)

Une application externe est une application informatique dont l'invocation est nécessaire à la réalisation de la tâche ou à l'exploitation des résultats générés avant de déclencher la tâche suivante ou de recommencer cette première. On tiendra compte de l'allocation de ressources, si l'application n'est pas uniquement informatique. Il faut différencier les outils (Tools), qui sont eux directement interfacés par le système Workflow, sans l'intervention d'une ressource du Système Workflow.

4.5.4 Concepts secondaires (*Wider Workflow Concepts & Terminology*)

Le paragraphe précèdent a introduit les concepts Workflow fondamentaux. Afin de présenter plus en détails le management de systèmes par Workflow, nous rappelons ici quelques définitions complémentaires.

4.5.4.1 Cas de procédure (*Process instance, Workflow Definition Instance*)

Un cas est une instanciation d'un modèle de procédure Workflow. Un cas est à la mise en œuvre d'un Workflow dans une situation spécifique, donc avec des paramètres particuliers. Par exemple, considérons un Workflow de remboursement des déplacements d'un représentant. Un cas de ce Workflow est le remboursement des frais de voyage de M. Duce, du 12 au 18 Avril 2006. L'instanciation d'un Workflow donne également lieu à l'instanciation des ac-

tivités du Workflow, il s'agit alors d'instances d'activités (Activity Instance), on parle également de tâche et d'activité (pour l'instance associée) [VAN DER AALST 98a].

4.5.4.2 Condition de transition (*Transition Condition*)

Une condition de transition est le critère de progression régissant le changement d'état d'une activité (étape de travail) ou le passage à l'activité (étape) suivante lors d'un cas d'exécution donné, qu'il s'agisse d'une procédure manuelle ou informatisée [WFMC11 99]. Une condition peut s'exprimer sous la forme d'une expression logique ou sous la forme d'un événement. Ces conditions sont soit intégrer dans les modèles de procédure, soit calculées par le moteur de Workflow au cours de l'exécution pour déclencher le modèle de procédure suivant adéquat.

4.5.4.3 Itinéraire, Transition, Enchaînement (*Route, Transition*)

Les itinéraires correspondent au contrôle de l'enchaînement des activités : il peut être choisi des routes séquentielles, parallèles, en disjonction ou en synchronisation entre les activités. Le contrôle est en général représenté sur un modèle Workflow à l'aide de contrôleurs de flux inspirés des opérateurs logiques associés à des contraintes de synchronisation portant sur les données pertinentes. La WfMC identifie 4 types basiques [WFMC03 95] répertorié Figure 15.

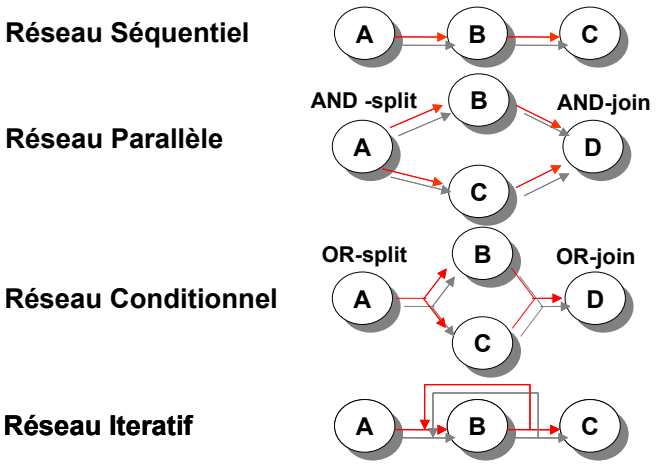


Figure 15 - Réseau de Processus

4.5.4.4 Bon de travail (*Work item*)

Un bon de travail est la représentation informatique du travail à effectuer par un acteur du Workflow dans le cadre d'une instance d'activité. Ce bon de travail peut également être l'interprétation d'un objet physique ou informatique se déplaçant dans la procédure Workflow dont les activités qu'il traverse le transforment successivement. Ce procédé peut être illustré par la transformation de matière première en produit fini.

4.5.4.5 Corbeille, Liste des bons de travail ou des tâches (*Worklist*)

La liste des bons de travail, contient la liste des bons de travail à exécuter par un acteur du Workflow sur une activité dans le cadre de son rôle, cette liste correspond a un échéancier d'événements à traiter pour la ressource en terme de modélisation & simulation.

4.6 Dimensions

Associé aux définitions précédentes, il a été défini une représentation tridimensionnelle des systèmes Workflow [VAN DER AALST 98a] (Figure 16).

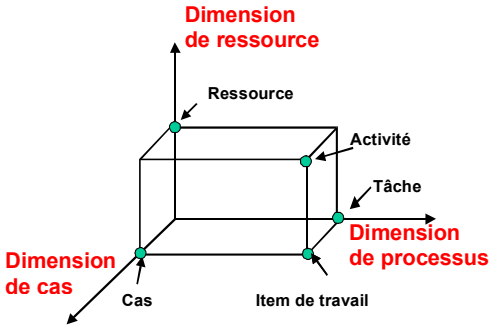


Figure 16 - Dimensions des systèmes Workflow

La Figure 16 donne une représentation tridimensionnelle d'un Workflow : avec une **dimension de cas**, une **dimension de processus** et une **dimension de ressource**. La dimension de cas représente l'instanciation du Workflow, chaque cas est un modèle à simuler et traiter individuellement. Les cas ne s'influencent donc pas directement. Ils peuvent éventuellement s'influencer indirectement via les ressources et les données. Le processus de Workflow est spécifié dans la dimension de processus, c'est-à-dire, la définition des tâches et de leur enchaînement. Enfin, dans la dimension de ressource, les ressources sont regroupées en rôles et en unités organisationnelles. En résumé, nous pouvons donc visualiser un certain nombre de termes Workflow dans la vue tridimensionnelle de la Figure 16. En particulier, une entité de travail est définie par la coordonnée cas + tâche et une activité par le cas + la tâche + la ressource. Cette représentation met en relief la gestion de Workflow comme le lien entre les cas, les tâches et l'organisation.

4.7 Description des Systèmes Workflow

Après avoir présenté la terminologie Workflow, il est maintenant possible de présenter le principe de fonctionnement des systèmes Workflow. A un haut niveau, ce type de système est composé de trois parties fonctionnelles [WFMC03 95]. La Figure 17 illustre un Système de gestion de Workflow et les relations entre ses différentes fonctions.

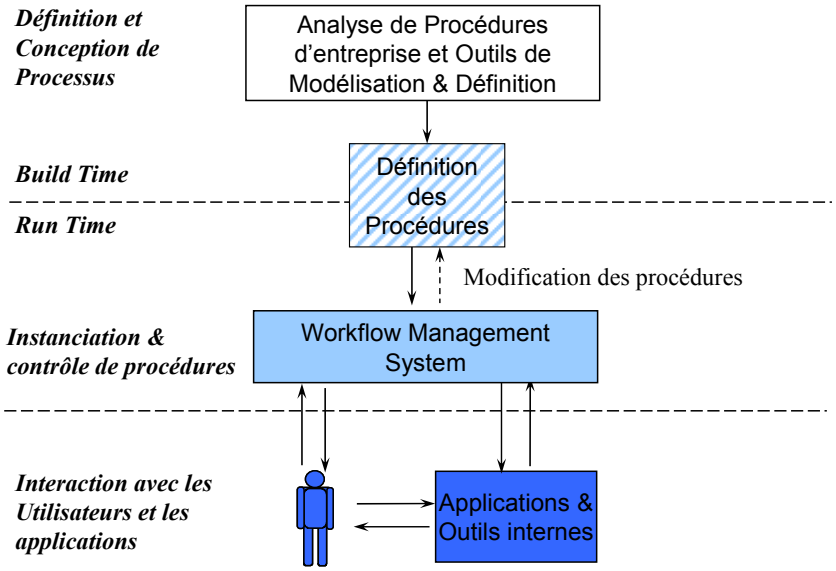


Figure 17 - Caractéristiques du système Workflow

4.7.1 Build time

Cette première phase permet la définition et la modélisation des procédures Workflow, elle est nommée *build time*. Elle est principalement composée d'un outil permettant la modélisation des Workflow dans un formalisme existant ou propriétaire à partir d'une analyse de procédures d'entreprises, il est à noter que les outils actuels préfèrent une description graphique. Un deuxième composant transcrit les modèles obtenus dans une définition des procédures « exécutable », c'est à dire compréhensibles par la partie chargée de l'exécution (simulateur du *run time*). Certains systèmes permettent en retour de la partie *run time* des modifications dynamiques de la structure de définition de procédures comme indiqué sur la Figure 17.

4.7.2 Run time

L'environnement d'exécution (ou *run time*) a l'entière gestion des modèles de Workflow établis dans la première partie. Cette gestion comprend l'exécution des Workflow avec un simulateur mais aussi la distribution des tâches aux rôles appropriés en cours d'exécution, la mise à disposition de l'ensemble des données et outils nécessaires, la supervision et le contrôle, etc.

Plus en détails, cette partie se décline en sous composants, introduits ci dessous :

Le Moteur de Workflow (*Workflow Engine*) qui est chargé de la Gestion des Procédures à travers la simulation de leurs évolutions.

Le gestionnaire des listes de tâches (*Worklist Handler*), chargé de distribuer les activités dans les listes des acteurs en fonction de leurs rôles.

Les listes de tâches (*Worklists*), qui sont les listes associées à chaque rôle dans lesquelles le moteur place les tâches à réaliser. Les tâches sont classées par priorité ou par date avec les données et les outils à utiliser de façon appropriée.

Les outils d'administration et de contrôle (*Workflow Process Monitoring*) suivent le déroulement des procédures Workflow, elles peuvent fournir l'état actuel des composantes du Workflow et donner l'ordre de modifier le modèle de procédures.

4.7.3 Utilisateurs, outils et applications

La troisième phase concerne l'ensemble des outils et des fonctions d'API (*Application Programming Interface*) qui permettent au système Workflow de s'interfacer avec des ressources humaines, des applications informatiques extérieures et d'autres systèmes Workflow.

4.8 Modèle de référence des systèmes Workflow

Le modèle de référence, Figure 18, présente l'architecture générale de l'environnement proposée par la WfMC, il identifie les interfaces couvrant cinq domaines de fonctionnalités entre le système Workflow et son environnement.

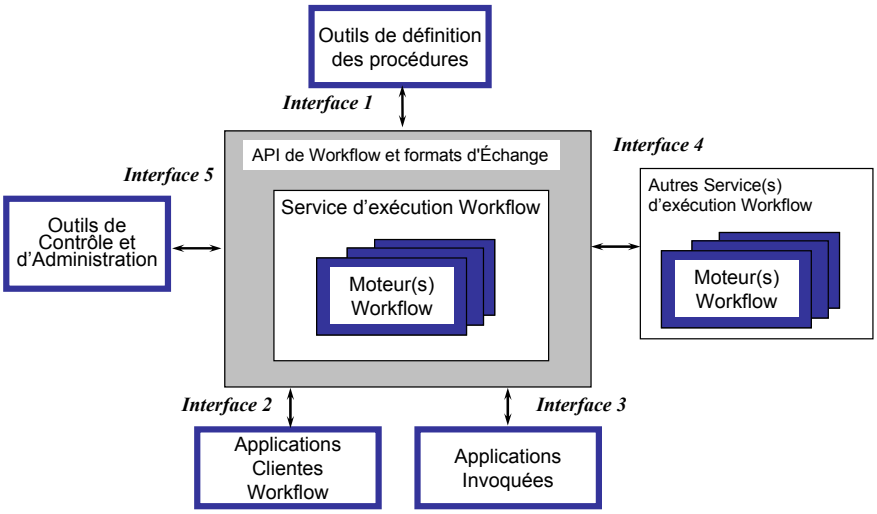


Figure 18 - Modèle de référence des systèmes Workflow

4.8.1 Interface avec les Outils de définition de procédures

Cette interface, située entre les outils de modélisation/définition et le logiciel de gestion du Workflow pendant l'exécution, est nommée interface d'import/export de définition de processus. Cette interface définit le format d'échange et d'appels des APIs, qui permettent l'échange d'informations de définition de procédures sur une variété de médias d'échange : physiques ou électroniques. Cette interface permet l'échange d'une définition de processus complète ou d'un sous-ensemble. Par exemple le changement de définition d'un ensemble de procédures ou plus simplement la modification des attributs d'une activité particulière dans une définition de procédures.

4.8.2 Interface avec les applications clientes Workflow

La liste des tâches (*Worklist*) à exécuter par une ressource est généralement définie et gérée par le service d'exécution du Workflow. Cette liste doit pouvoir déclencher des appels à des applications clientes diverses et des ressources. La solution retenue pour respecter la susdite exigence, consiste à encapsuler la variété d'application qui peut être utilisée derrière un jeu standard d'API (le WAPI *Workflow Application Programming interface*). Ce jeu permet ainsi d'utiliser une communication standardisée entre les applications clientes, le moteur de

Workflow et les Worklist, indifféremment de la nature de l'implémentation réelle des produits clients.

4.8.3 Interface avec les applications invoquées

Il est évident que le système Workflow ne peut pas intégrer l'invocation automatique de toutes les applications qu'il peut être amené à utiliser pendant l'exécution d'un Workflow. Par exemple les applications dont les données sont fortement typées. Dans ce cas un composant externe supplémentaire, nommé agent d'application, est ajouté, il est chargé de la traduction des informations dans un format compréhensible par le standard WAPI.

Dans le cas le plus simple, l'invocation d'application est traitée localement par un moteur de Workflow, mais les applications invoquées peuvent être utilisées par plusieurs moteurs de Workflow et peuvent se situer sur des machines distantes, il convient donc de définir un format commun d'utilisation des ces applications entre les Workflow dans le but de communiquer correctement et de synchroniser l'appel à ces applications.

4.8.4 Interface avec les autres Workflow

Un des objectifs de la normalisation dans la définition de Workflow est de pouvoir transmettre des *WorkItem* entre deux systèmes Workflow conçus par des concepteurs de systèmes Workflow différents. Trois principaux types d'interopérabilité ont étés identifiés :

Workflow chaînés : La dernière activité d'un Workflow A doit pouvoir fournir un item à la première activité d'un Workflow B.

Workflow hiérarchiques : une activité d'un Workflow A doit pouvoir être vu comme un Workflow B.

Workflow *Peer to Peer* : Une procédure globale est composée d'activités gérées en partie par un Workflow et en partie par un autre Workflow, sans système de supervision de la procédure complète.

Workflow Synchronisés : Deux Workflow s'exécutent en parallèle et doivent pouvoir se synchroniser sur certaines activités.

Pour résumer, il est possible d'identifier deux aspects principaux nécessaires à l'interfonctionnement de Workflow :

- L'interprétation commune de la définition de procédures (ou d'un sous-ensemble).
- L'appui pendant l'exécution de l'échange des divers types d'information de contrôle et le transfert des données appropriées et/ou d'applications entre les services d'exécution Workflow différents.

4.8.5 Interface avec les outils de contrôle et d'administration

L'objectif de cette interface est de permettre à un logiciel de Monitoring de Workflow de s'interfacer avec plusieurs Workflow différents et ainsi regrouper la supervision d'un ensemble de systèmes Workflow dans un logiciel.

L'interface 5 permet à une application de gestion indépendante d'interagir avec des Workflow de différents domaines. L'application de gestion peut aussi se charger d'autres fonctions de gestion, au-delà de celles-ci. Par exemple, elle peut aussi gérer des définitions de procédures de Workflow, agissant comme un dépôt d'information commun à plusieurs systèmes et distribuant des définitions de processus aux divers Workflow via des opérations au travers de leurs interfaces 1.

Malgré cela, des scénarii d'implémentations moins modulaires sont aussi envisageables; par exemple l'application de gestion peut être une partie intégrante du service d'exécution.

4.9 Classification des systèmes Workflow

Il n'existe pas de classification commune des systèmes Workflow dans la littérature, reconnue par l'ensemble de la communauté Workflow [VAN DER AALST 02]. Ceci étant essentiellement dû au nombre important de critères de classification qu'il est possible de retenir. En effet, les spécialistes adoptent différents points de vue par rapport à la notion de Workflow, les critères qui en découlent varient donc en fonction de leurs perceptions des caractéristiques présentées par ces systèmes Workflow. Ainsi, il existe plusieurs classifications, permettant de sélectionner un outil de gestion de Workflow avec différents « éclairages » sur le sujet.

Malgré ce manque d'unité, la classification proposée par [McCREADY 92] est assez répandue dans la littérature, elle est reprise par bon nombre d'auteurs [VAN DER AALST 98a], [GEORGAKOPOULOS 95]. Elle propose de distinguer quatre catégories de Systèmes Workflow. La Figure 19 présente ces différentes classes selon deux axes : Approche et Structure.

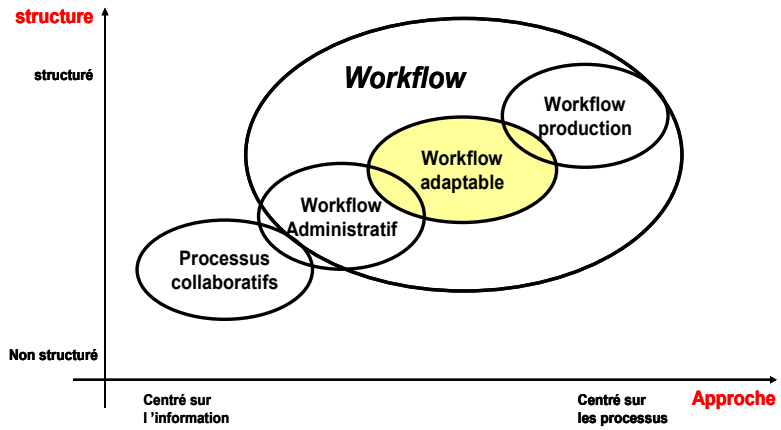


Figure 19 - Différentes classes des systèmes Workflow

4.9.1 Processus collaboratifs

Cette première classe est axée sur la communication et sur le partage d'information. Les systèmes collaboratifs sont définis pour supporter le travail en groupe, dans le cadre de la conception, de la gestion de projet ou de la résolution de problèmes faisant appel à plusieurs niveaux d'expertise. Ces systèmes permettent de réunir les intervenants d'un projet autour d'un objectif commun, les clients de la procédure y étant souvent eux-mêmes directement

associés, les logiciels employés sont le plus souvent des *groupwares*[27]. Les tâches des procédures gérées sont le plus souvent complexes et leur réalisation implique l'intervention de ressources aux compétences très spécifiques pour une forte valeur ajoutée. D'un autre coté, l'enchaînement des activités des procédures à traiter est faiblement structuré et peu répétitif. De part la faible structure de ces processus, ils ne font pas partie ou se situe à la frontière de ce que l'on considère comme la « sphère » Workflow comme le précise la Figure 19.

4.9.2 Workflow administratif

Les systèmes Workflow administratifs (*General Purpose Workflow Management Systems*) ont pour objectif de décharger les ressources d'une entreprise des tâches administratives. En effet ces procédures sont répétitives, fortement prédictibles et les règles d'enchaînement des tâches sont très simples et clairement définis ; ces procédures sont donc aisément automatisables, évitant ainsi un travail fastidieux où peuvent naître des erreurs souvent humaines. Les systèmes Workflow administratifs permettent de lier à une tâche administrative, les documents et les informations nécessaires à la réalisation de cette tâche par un acteur humain. Ces systèmes gèrent également le routage des documents et le remplissage de formulaires. La gestion par Workflow de procédures administratives permet un gain de l'ordre de 5% à 10% en termes de productivité et de 30 à 90% en termes de délais [ADER 99]. Enfin une dernière raison de l'automatisation de ce type de procédures en Workflow provient du fait que ces procédures possèdent une structure statique et ne sont donc pas souvent assujetties à modifications car elles possèdent une longue durée d'utilisation [SCHEER 97].

4.9.3 Workflow de production

Les systèmes Workflow de production impliquent des procédures prévisibles et assez répétitives. Leurs principales différences avec les Workflow administratifs résident dans la complexité des tâches et de la structure des procédures, dans leur capacité à faire appel à des informations provenant de systèmes d'information variés et dans l'enjeu que représente leur réussite. En effet, la procédure Workflow correspond directement au travail effectué par l'entreprise. En d'autres termes, la performance de l'entreprise est directement liée à l'exécution de la procédure managée par le Workflow. On dit dans ce cas qu'il est *mission critical*[28] [INCONCERT 97]. C'est par exemple le cas des organismes financiers, des compagnies d'assurances, des usines de production manufacturières. La réalisation des procédures est donc associée à une forte valeur ajoutée et un volume d'informations traitées important.

La complexité des procédures traitées est également due à la répartition de leurs activités sur plusieurs sites. Dans ce cas, les tâches exécutées nécessitent souvent l'interrogation de plusieurs systèmes informatiques, hétérogènes et distribués. Il est donc nécessaire que les systèmes Workflow de production fournissent un ensemble d'outils ou de fonctions d'API per-

[27] Collecticiel : classe de logiciels prévus pour être exploités de façon concurrente, sur un même projet.

[28] Un système est dit critique lorsque : les vies de personnes sont tributaires de son fonctionnement ou le coût économique d'un dysfonctionnement est catastrophique.

mettant de se connecter à plusieurs systèmes. Enfin, même si les procédures traitées sont assez répétitives, elles sont susceptibles d'être modifiés plus souvent que les procédures administratives, car associées à la modification des objectifs du métier. Ces modifications peuvent par exemple avoir lieu dans le cadre d'une restructuration de BPR[29] ou d'un CPI[30].

Les systèmes Workflow de production doivent donc pouvoir évoluer. Par ailleurs, l'exécution de certaines procédures ne peut pas toujours se poursuivre de manière automatisée, suite à l'occurrence d'un ou de plusieurs événements qui font aboutir le système dans un état particulier. Dans ce cas, il est nécessaire de faire intervenir des acteurs humains pour la prise de décision. Pour ce faire, le système Workflow de production peut faire appel à un autre système, de type collecticiel ou un autre système Workflow ad hoc, qui servira d'interface pour l'exécution dirigée par un acteur humain de la suite de la procédure. Ce type de Workflow est dit « composite » [EDER 96]. Enfin, dans la littérature, les systèmes Workflow de production sont également appelés *case-based* [VAN DER AALST 98a].

4.9.4 Workflow adaptable ou Workflow ad hoc

L'impossibilité pour les systèmes de gestion de Workflow traditionnels de traiter les différents changements dynamiques dans les flux de travail est une limite à dépasser. A ce titre, il a été introduit les concepts de Workflow adaptable (*adaptive Workflow)* [VAN DER AALST 98c] et de Workflow ad hoc [VOORHOEVE 97]. La nuance entre ces deux nouveaux termes provient du fait qu'ad hoc désigne un acte spécialement fait pour un objet déterminé alors qu'adaptable prévoit un changement définitif de la procédure.

Les Workflow ad hoc se situent à la frontière gauche de la représentation Figure 19 dans la « sphère » Workflow adaptable. Ils régissent des procédures dont la structure est déterminée pendant l'exécution en fonction des décisions humaines prises suite à la réalisation d'une tâche, plus concrètement, la structure se construit par pas en suivant le rythme de l'exécution. En effet, la réalisation d'une procédure non structurée peut impliquer à chaque fois l'exécution d'un nouvel enchaînement des tâches, voire la création de nouvelles tâches. Il n'y a pas a priori de persistance de l'enchaînement de ces tâches.

Les Workflow adaptables sont, quant à eux, des supports comparables aux Workflow de production classiques possédant une structure préétablie, mais pouvant traiter certains changements de structure « en ligne ». Ces changements peuvent aller des changements individuels/ad hoc (gestion d'exception), c'est à dire d'un aiguillage pour déterminer l'activité suivante, jusqu'à la reconception par BPR de processus [VAN DER AALST 98d]. En conclusion, ils ont une action globale pouvant inclure la définition du Workflow ad hoc

Il est intéressant de classifier les différents changements possibles par un Workflow adaptables, dans le but de mieux les anticiper [SADIQ 99]. Les changements sont envisageables selon plusieurs perspectives : la ressource, le contrôle, la procédure, la tâche et le système

[29] Business Process Reengineering 'Reconfiguration des Processus Métier'

[30] Continous Process improvement.

[VAN DER AALST 98d]. Dans la suite de l'étude seul l'aspect procédure sera développé, c'est en effet la perspective dominante du management par Workflow et elle comporte un aspect important : les changements dynamiques. Nous présentons ci-dessous les différents types de changements envisageables.

4.9.4.1 Le changement individuel (ad hoc)

Les systèmes Workflow ad hoc sont utilisés pour l'exécution de processus non structurés ou peu structurés (sujets à changement). Un processus peu ou non structuré est un processus dont l'ordre et le temps exact de réalisation des tâches ne sont pas établis au préalable et/ou peuvent être modifiés pendant l'exécution. Les choix de routage et la nature des tâches sont décidés au fur et à mesure de l'exécution. Par conséquent, un processus non structuré propose un objectif immuable, mais pouvant être atteint de différentes façons. Certaines situations rencontrées pendant le « *run time* » nécessitent donc des dérivations ad hoc dans la procédure, éventuellement planifiée, comme le proposent [HAN 98], telles que :

Le raffinement Dynamique :

Dans certains cas, il est impossible ou peu pratique de définir une spécification complète du modèle de Workflow. En raison de l'indisponibilité d'une spécification complète, le raffinement dynamique peut être nécessaire pendant le *« run time »*, c'est-à-dire que certaines tâches ne seront complètement et définitivement spécifiées qu'en *« run time »*. Le même raisonnement peut être appliqué pour définir les ressources exigées pour l'exécution d'une tâche.

La participation d'Utilisateurs :

Au lieu d'être des contrôleurs passifs, certains utilisateurs d'un système de Workflow doivent être traités comme des « propriétaires d'une tâche ou d'une procédure » [HAN 98]. L'approche de ces systèmes est souvent de type « pull », c'est-à-dire que leurs utilisateurs doivent les interroger pour connaître l'état du processus et en déduire leurs tâches ; par opposition, les autres types de systèmes, possède eux une approche plutôt de type « push », où les utilisateurs sont informés par le système des travaux qu'ils ont à traiter [GEORGAKOPULOS 95]. Techniquement, la métaphore utilisée dans ce type de système est celle du « dossier » [WAINER 95]. Les utilisateurs font circuler un dossier virtuel dans lequel sont « placés » des documents et des données électroniques. Chaque utilisateur en possession du dossier décide du prochain destinataire. Le processus décisionnel de l'utilisateur doit être considéré dans l'exécution de processus de Workflow.

Adaptation aux événements externes :

Des événements non pris en compte par le modèle de Workflow, y compris certains stimuli externes, l'intervention d'utilisateurs, les temps morts, etc., doivent être traités correctement pour résoudre les problèmes du monde réel et faciliter la communication entre des procédures Workflow différentes. En outre, une fois qu'une communication inter-Workflow a lieu, les utilisateurs ou les propriétaires de procédures doivent être capables de répondre à ces

événements en raffinant dynamiquement leur procédure ou en modifiant la tâche actuelle et/ou les interdépendances de tâches.

Situation d'échec :

Un défaut système, des conflits de ressource et des fausses opérations peuvent causer des erreurs et des difficultés dans l'exécution d'une procédure Workflow. Les mécanismes pour traiter les situations d'erreur sont donc très importants pour assurer l'amélioration des processus de Workflow.

4.9.4.2 Le changement structurel (évolution)

Ces changements sont souvent une réaction pour s'adapter à un changement d'environnement dû au contexte concurrentiel très dynamique ou au besoin d'adaptation aux progrès technologiques, au travers la parution de nouveaux logiciels ou de nouvelles versions [HAN 98].

Ces changements sont souvent le fruit d'un travail de BPR.

Après une telle modification, il existe plusieurs possibilités [VAN DER AALST 98d] et [SADIQ 99] d'intégrer les cas existants dans la nouvelle procédure, contrairement aux changements ad hoc qui restent un traitement d'exceptions et sont gérés individuellement.

Première possibilité : Redémarrer « *restart* » « *abort* »

Les cas en cours de traitements dans l'ancien processus sont remis à zéro et redémarrés dans le nouveau processus au lancement du nouveau système.

Deuxième possibilité : Parallèle « *proceed* » « *flush* »

Le système conserve en parallèle l'ancien et le nouveau processus le temps de l'exécution des cas en cours sur l'ancien processus.

Troisième possibilité : Transférer « *transfer* » « *migrate* »

Ce changement n'affecte pas le traitement des cas qui sont transférés directement dans le nouveau processus dans l'état actuel de leur déroulement.

4.10 Modélisation des processus Workflow

La modélisation est une activité qui précède toute décision ou formulation, elle permet de représenter la description du système réel. Tout comme un système informatique, le système Workflow comporte un certain nombre d'aspects à modéliser que nous allons présenter ci-dessous.

4.10.1 Aspects à modéliser

4.10.1.1 L'aspect fonctionnel :

L'aspect fonctionnel concerne l'identification des activités des processus que l'on souhaite modéliser. Il est important de comprendre qu'il ne s'agit pas uniquement d'identifier les fonctions des différents départements d'une organisation mais bien de distinguer les activités composant un processus. La modélisation fonctionnelle doit également permettre d'établir la

hiérarchie des activités, c'est-à-dire d'exprimer de possibles décompositions en termes de sous-processus. Enfin, le modèle fonctionnel doit aussi représenter le flux de données associées aux activités et les interdépendances de données entre les activités (*data flow*).

4.10.1.2 L'aspect comportemental :

L'aspect comportemental est un aspect primordial du Workflow puisqu'il correspond à la dynamique du processus. Le comportement s'exprime par la modélisation d'un contrôle de flux entre les activités. Ce dernier permet d'indiquer la chronologie de l'exécution des activités, leur flux (séquentiel ou parallèle), les points de synchronisation entre activités ou au contraire, les points de disjonction. De plus, le modèle comportemental doit représenter les évènements qui permettent de déclencher les activités. Notons pour souligner l'importance de ce modèle, qui permet l'exécution du Workflow. L'aspect comportemental est également appelé aspect de coordination.

4.10.1.3 L'aspect informationnel (donnée) :

Cet aspect concerne l'ensemble des informations et des données qui sont associées aux activités. Le modèle informationnel, souvent négligé lors de l'implémentation d'un Workflow [JOOSTEN 96], décrit en détail les relations qui existent entre les données, leur type et leur structure.

4.10.1.4 L'aspect organisationnel :

Comme son nom l'indique, la partie organisationnelle concerne la description de l'organisation des acteurs de l'entreprise. Le modèle organisationnel peut refléter fidèlement l'organigramme de l'entreprise, c'est-à-dire la décomposition hiérarchique de celle-ci en départements et services ou bien décrire des unités organisationnelles dans lesquelles on identifie des acteurs. Selon la méthode choisie, la description est plus ou moins détaillée et permet d'établir des liens hiérarchiques entre les acteurs ainsi que des relations entre unités organisationnelles ou départements. Toutefois, quelle que soit la méthode retenue, la description des rôles associés aux différentes activités reste invariante. Les rôles créent l'interface entre le modèle organisationnel et les modèles représentant les activités.

4.10.2 Modélisation de Workflow adaptable

Dans le cas particulier de la modélisation des Workflow adaptables, plusieurs approches sont proposées [HAN 98].

4.10.2.1 Approche Meta modèle :

Cette approche emploie un méta modèle pour déterminer la structure et les types de composants constitutifs; elle définit un jeu de primitives pour transformer un modèle Workflow générique ou une instance de modèle.

4.10.2.2 L'approche de point ouvert *Open-point* :

Elle définit des points spéciaux dans un modèle de Workflow, où l'adaptation peut être faite. Cette approche inclut la mise à disposition de choix multiples pour les utilisateurs, l'allocation dynamique de certaines ressources aux tâches en *run time*, ou une interface ouverte par laquelle une « dernière modélisation » peut être réalisée. La « dernière modélisation » signifie que, pendant l'exécution des modèles complets, certains sous modèles peuvent être définis dynamiquement pour une utilisation immédiate.

4.10.2.3 Approche synthétisée :

Il semble que les deux approches puissent être employées complémentairement pour la définition de Workflow adaptable. Pour cela, les deux hypothèses suivantes doivent être prises en compte :

Tout d'abord, il parait nécessaire de distinguer les évolutions de procédures Workflow et les changements ad hoc dynamique dans une procédure. De plus, une séparation claire est faite entre le *build time* et le *run time* en termes de Workflow. Il parait impératif de faciliter le franchissement bilatéral de cette frontière pour rendre modélisables les Workflow adaptables.

En synthèse, il est souhaitable de définir la frontière entre les changements ad hoc et les évolutions du Workflow. Il semblerait que ces changements devraient être testés sur une instance de modèles puis si nécessaires propagés au niveau global pour une modification persistante.

4.10.3 Techniques et outils de modélisation de Workflow

Associés aux aspects à modéliser définit précédemment, un certain nombre d'outils de modélisation peuvent être employés pour décrire le comportement des flux de travail comme le précise [VAN DER AALST 97a] [VAN DER AALST 97b].

Il existe un certain nombre de techniques de modélisation des procédures, tels que :

Pour la représentation en terme de fonctions, les diagrammes de flots de données DFD[31], ISAC[32], SADT[33] semblent tout a fait indiqués.

Pour l'aspect comportement, les diagrammes état-transition (par extensions : *state charts*) semblent convenir plus particulièrement. Mais aussi les organigrammes (*Flowcharts)* qui illustre les pas dans une procédure. En visualisant les procédures, un organigramme peut rapidement aider à identifier des goulots d'étranglement ou des inefficacités dont la procédure peut être rationalisée ou améliorée.

Concernant l'aspect représentation des données, on pourra utiliser par exemple un Modèle Conceptuel de Données de la méthode MERISE [TARDIEU 85] ou un modèle de classes

[31] Data Flow Diagram

[32] Information Sharing and Analysis Center

[33] Structured Analysis and Design Technique

d'une méthode Objet UML[34]. Le modèle informationnel peut également représenter le flux des informations, c'est-à-dire leur circulation et les différents états qu'elles peuvent prendre ainsi que les dépendances hiérarchiques.
Il existe également des techniques de diagramme spécifiques aux concepteurs de logiciels utilisées dans les outils de simulation WfMS. Ces outils de modélisation et simulation spécifiques intègrent un outil modélisation, le plus souvent dans un langage graphique et un simulateur également spécifique se basant sur un formalisme particulier après transformation dans ce dit format simulable, par exemple un automate à états finis.
Mais ce sont certainement les réseaux de Petri de haut niveau qui sont les plus utilisés à l'heure actuelle pour représenter et simuler les comportements des procédures d'un Workflow autour des nombreux travaux de Van der Aalst [VAN DER AALST 98b].
Cependant, jusqu'alors pas, ou peu, de travaux de spécification et de modélisation de la partie processus d'un Workflow on été réalisés en utilisant la clarté et la précision du formalisme DEVS. Ce formalisme offre néanmoins un aspect modulaire hiérarchique important dans la définition d'un Workflow. De plus leur simulation par événements discrets permet une exécution rapide. Nous reviendrons sur ce point dans le chapitre 4.

4.11 Outils logiciels pour décrire, analyser et valider un Workflow

Pour représenter et analyser les Workflow, il a été développé un certain nombre de logiciels. L'ensemble des logiciels de Workflow dénombre aujourd'hui plus de deux cents logiciels sur le marché et de plus en plus entreprises envisagent l'utilisation de tels outils [VAN DER AALST 02]. Ce choix important de logiciels provient de la jeunesse du domaine et du manque de standards conceptuels formellement décrits.

Dans le but de donner une impression de la génération actuelle de Systèmes de gestion Workflow, nous présentons ici brièvement cinq d'entres eux, trois utilisés dans le domaine industriel Cosa, Staffware [DESEL 00] et Workey, ainsi que deux plus orientés pédagogie provenant du monde universitaire : Yawl et Yasper. Les outils universitaires permettent de modéliser et d'analyser les Workflow sur les bases formelles des réseaux de Petri [VERBEEK 01]. La Figure 20, ci après, permet de résumer les caractéristiques saillantes des logiciels présentés.

4.12 Conclusion Workflow

Depuis quelques années, l'offre de logiciels commerciaux et open source a explosé avec à l'heure actuelle une offre segmentée par type de Workflow à mettre en œuvre. L'arrivée de logiciels *open-source* écrits en Java atteste de l'effervescence du domaine. [PEREZ 05] propose une liste de tous ces environnements, contenant notamment YAWL. Il est donc évident que ce type de système va jouer un rôle majeur dans un futur proche.

[34] Unified Modeling Language

Staffware (Tibco) [STAFFWARE 01]	25% du marché en 2000, Staffware comporte les composants suivants : - Workflow Graphique Definer (GWD) : Outil de définition de processus. ne permet aucune forme d'analyse - Designer de Forme Graphique (GFD) : Utilisé pour définir l'interface qui est présentée à l'utilisateur final ou, en cas de tâche automatique, l'interface est présentée à la l'application externe. Ne permet pas de représenter tous les enchaînements possibles pour les activités d'un Workflow - Manager de File d'attente de Travail (WQM). outil client utilisé pour offrir le travail aux utilisateurs finaux - Staffware Serveur (SS) : Partie exécution du Workflow - Staffware Managers d'Administration (SAM) : Consiste en jeu d'outils pour soutenir les administrateurs de Workflow. Les outils suivants sont inclus : le manager d'utilisateur, le manager de secours, le manager de table, le manager de cas, le manager de liste, le manager de réseau - Protocole d'audit (AT) : Utilisé pour contrôler l'exécution de cas individuels.
Yawl (Faculty of Information technology Queensland University Australia) [YAWL 05]	- YAWL est basée sur les modèles de Workflow - Réseaux Petri. - Le système de YAWL inclut un moteur d'exécution - Un éditeur graphique - Conforme aux licences open-sources.
Cosa (Software-Ley) [COSA 02]	- Basé sur les réseaux de Petri - Permet la modélisation, le contrôle et l'exécution de WF - Ne permet pas la vérification, la simulation est détaillée et difficile à comprendre pour un utilisateur non expérimenté
Yasper (University of TU Eind-hoven and Deloitte Netherlands) [YASPER 05]	- Basé sur les réseaux de Petri - Modélisation graphique - Simulation animée - Extensions simples pour procédures de Workflow - Disponible librement - Des rôles (des ressources) peuvent être assignés aux éléments - Simulation manuelle affichée dans une interface spéciale - Simulation automatique - Hiérarchies de sous processus - Yasper est écrit en .NET, utilisation de Fenêtres et de Formulaires (aucun appui de Linux)
Workey (c-log France) [CLOG 06]	- Modélise, documente, diffuse les processus de l'organisation de l'entreprise en Intranet - Connecteurs définis et présents pour chaque jonction du flux et définis explicitement. - Permet de représenter les processus Workflow de manière hiérarchique - Un éditeur graphique - Donne la possibilité de gérer des objets informatiques en fin de conception - Workey se base sur un système Lotus (génération orienté lotus note) - Nécessite une installation lourde

Figure 20 - Tableau des fonctionnalités de logiciels de modélisation Workflow

5 Conclusion

Ce chapitre a présenté un état de l'art utile aux fondements des travaux développés dans le cadre de cette thèse. Nous avons tout d'abord rappelé les principales notions relatives à la théorie de la modélisation et de la simulation. Nous nous sommes ensuite centrés sur les formalismes DEVS et G-DEVS comme étant les bases de spécifications formelles des modèles utilisés par la suite. Puis, nous avons présenté les principes relatifs à la simulation distribuée, notamment la norme HLA reconnue comme un moyen de rendre interopérables des simulations. Enfin nous avons introduit les systèmes de gestion de procédures d'entreprises nommés Workflow.

L'ensemble de ces concepts sert de base aux développements des chapitres suivants.

Chapitre 2. Modélisation & Simulation G-DEVS / HLA

1 Introduction

Deux des enjeux actuels du domaine de la modélisation & simulation sont la performance et la capacité à pouvoir réutiliser et distribuer les composants. Ces aspects exigent le développement de structures de modélisation & simulation modulaires et de protocoles de simulation distribuée efficaces.

Nous définissons dans ce chapitre un simulateur conceptuel de modèles à événements discrets, dotés d'une modularité issue de la représentation de modèles hiérarchiques. Ce simulateur s'inspire de la structure définie par [ZEIGLER 00] mais effectue une mise à plat des modèles dans l'objectif d'un gain de performance.

Afin de répondre à l'enjeu de distribution des modèles, il est souhaitable de créer des modèles décomposables en sous modèles distribués. Pour que ces composants de simulation puissent être implémentés sur plusieurs ordinateurs avec différents systèmes d'exploitation, il est nécessaire de définir un protocole de communication qui permet d'intégrer ces différents éléments dans une simulation globale distribuée. Nous proposons un simulateur distribué conceptuel de modèles mis à plat et montrons que la norme HLA permet de mettre en œuvre les concepts précédents en proposant des méthodes d'échanges de messages entre les simulations locales, et la synchronisation des différentes entités. La solution proposée consiste à créer des simulations à événements discrets distribuées communiquant par l'interface spécifiée dans HLA. Nous proposons, en particulier, de faire communiquer les différentes simulations locales directement au travers du RTI HLA sans utiliser un composant spécifique pour la gestion du temps [ZEIGLER 99]. Chaque simulation locale devra intégrer des algorithmes de synchronisation, issus de la mise en oeuvre de HLA. Ces algorithmes permettent d'assurer le respect du déterminisme et de la causalité. Cette solution présente, de plus, un nouvel algorithme d'intégration DEVS/HLA utilisant le Lookahead HLA. Enfin, nous introduisons deux méthodes de calcul du Lookahead permettant de prendre en considération la dynamique des modèles G-DEVS dont les fonctions « durée de vie » sont dépendantes d'au moins une variable d'état.

2 Simulation séquentielle de modèles G-DEVS

B.P. Zeigler a défini la spécification d'une structure de simulation conjointement à sa définition de modèles atomiques et couplés hiérarchiques DEVS. Cette structure de simulation exploite les aspects hiérarchiques et modulaires provenant de la représentation des modèles. Cette spécification est générique et ne prédispose pas d'un contexte particulier d'implémentation. Enfin, cette structure s'applique également aux modèles G-DEVS.

2.1 Simulation des modèles DEVS

Le formalisme DEVS donne une spécification d'un système (exemple : par un modèle couplé DEVS), pour laquelle on définit des valeurs initiales des variables d'états et des trajectoires d'entrée pour tous les ports d'entrée du modèle. A partir de ces données, il doit être précisé comment générer les états suivants et les trajectoires de sorties. Pour cela, des algorithmes de simulation sont définis dans un simulateur abstrait qui les implémente.

Il existe plusieurs simulateurs abstraits correspondant à des modèles DEVS particuliers et à des extensions de DEVS. Dans la section suivante nous rappelons le simulateur abstrait de DEVS basiques défini dans [ZEIGLER, 00].

2.1.1 Rappel Simulateur conceptuel DEVS

[ZEIGLER 95] présente un modèle de données définissant les différentes classes de la structure de modèles et de simulation, ainsi que le détail des attributs pour chacune de ces classes. Ce diagramme d'objets est rappelé sur la Figure 21.

En ce qui concerne la structure de modélisation, chaque classe *modèle* possède un attribut nommé parent qui définit son parent dans la représentation structurelle hiérarchique. Elle possède également un attribut qui lui associe un objet processeur pour la simulation. Enfin, elle possède un champ contenant les ports d'entrée et de sortie pour assurer le couplage, la réception et l'émission des messages.

Chaque modèle atomique est supporté par une classe *modèle atomique*. Ces modèles héritent de la classe *modèle* exposée ci dessus. En complément des fonctions reçues par héritage, ils possèdent les fonctions classiques du formalisme DEVS définissant les transitions internes/externes, les sorties et la durée de vie. Chaque *modèle atomique* possède par héritage un processeur nommé *simulateur* associé.

Les modèles DEVS couplés sont supportés par une classe *modèle couplé* héritée de la classe *modèle*. Chaque classe de modèle couplé est associée à sa propre classe de processeur nommée *coordinateur*. La classe *modèle couplé* possède, de plus, des attributs décrivant les relations d'influences (ou de couplage) entre ses modèles composants (appelés modèles fils).

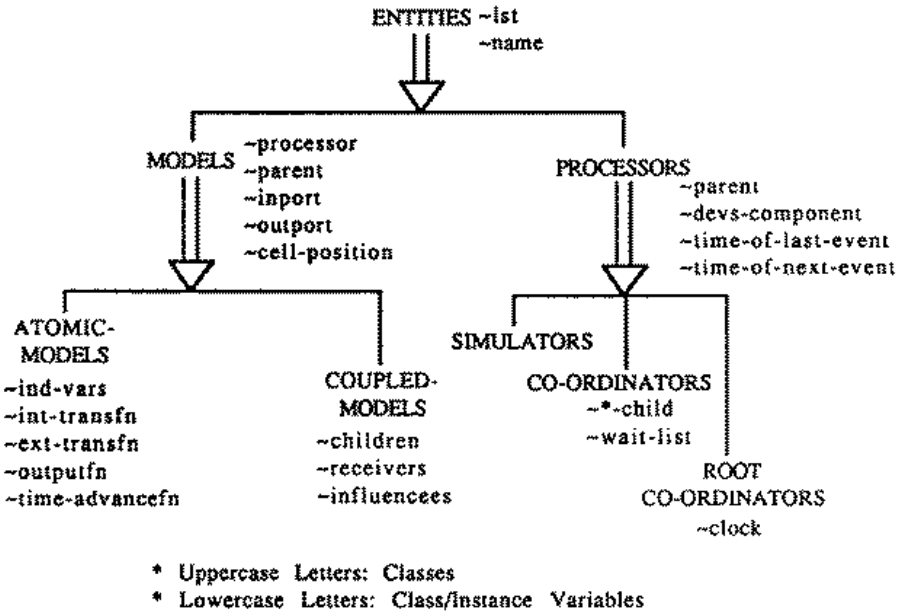


Figure 21 - Classes de modèle et simulateur [ZEIGLER 95]

Si l'on considère maintenant la structure de simulation (rappelée sur la Figure 21), les classes du simulateur abstrait héritent de la classe *processeur*, et possèdent donc toutes des attributs 'date de prochain événement' et 'date de dernier événement traité' ainsi qu'un attribut 'parent' [ZEIGLER 95]. La classe *coordinateur* possède un attribut 'liste d'attente des messages transmis aux successeurs' et un attribut définissant ses processeurs fils. La classe *coordinateur racine* possède un unique attribut 'horloge'. Nous présenterons plus précisément ces différents composants dans les sections suivantes.

Tous les processeurs adhèrent à un protocole d'échanges de messages qui permet de coordonner la simulation [ZEIGLER 98a]. Cet échange pourra être conçu et développé de plusieurs façons car la spécification du simulateur conceptuel défini ce qui doit être fait mais pas comment cela doit être fait.

La Figure 22 représente la structure hiérarchique de simulation calquée sur la structure de modélisation introduite dans le premier chapitre. Notons l'ajout, dans la structure de simulation, d'un composant racine au sommet de la hiérarchie qui a pour fonction d'initier les cycles de simulation et de gérer le temps simulé.

Le premier intérêt de cette représentation hiérarchique de la structure de simulation est basé sur la possibilité de transformer directement un modèle hiérarchique en code de simulation exécutable. Un autre avantage de cette représentation est lié au fait qu'elle supporte l'interopérabilité des formalismes. En effet, comme la simulation s'exécute par échanges de messages, un coordinateur peut communiquer avec des simulateurs hétérogènes à la seule condition qu'ils communiquent aussi par passage de messages et qu'ils puissent être vus, par le coordinateur avec lequel ils sont connectés, comme des simulateurs de modèles DEVS basiques. Cette faculté à interopérer est nommée le concept de *DEVS-bus* [ZEIGLER 98a].

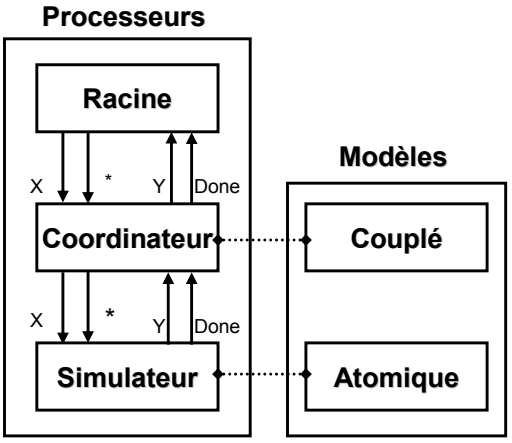


Figure 22 - Correspondance entre modèles et simulateur hiérarchique [ESCUDE 00]

2.1.1.1 Messages échangés

Nous revenons à présent sur la notion de « messages » présentée dans le premier chapitre. Nous pouvons distinguer principalement 3 types de messages échangés par les coordinateurs et les simulateurs (cf. Figure 22), qui implémentent la notion d'événement.

Lors du lancement de la simulation, un message d'initialisation (i, t), également appelé *Imessage*, est envoyé depuis le coordinateur racine vers tous ses fils dans le but d'initialiser les états actuels des modèles et de récupérer les informations de prochaines transitions internes planifiées. Ce type de message n'est pas représenté sur la Figure 22 et n'est pas comptabilisé dans les types de messages car il ne comporte qu'un intérêt dans la mise en œuvre du simulateur. Les événements de prochaines transitions internes sont transmis dans des messages (*, t), également appelés **messages*. En retour du traitement d'une transition interne, les simulateurs peuvent émettre un événement de sortie (y, t), également appelé *Ymessage*, qui remontera jusqu'au coordinateur de plus haut niveau et sera transformé en message d'entrée. Les messages d'entrée notés (x, t), également appelés *Xmessages,* sont créés pour chaque événement reçu sur un port d'entrée du modèle, ou pour chaque *Ymessage* provenant d'un sous modèle influenceur après transformation en fonction des relations de couplage.

2.1.1.2 Simulateurs

Le comportement dynamique des modèles DEVS est exécuté par les fonctions des simulateurs associés aux modèles atomiques. Lors de la réception des événements, ils activent les fonctions définies dans le modèle. Ces simulateurs manipulent notamment deux variables temporelles nommées tl[35] et tn[36]. La première variable tl mémorise la date du dernier événement traité et tn la date du prochain événement à traiter. Il peut être déduit de ces deux variables la durée de vie de l'état actuel, notée ta(s)[37], où s est l'état actuel du modèle.

$$tn = tl + ta(s)$$

[35] Time of Last event
[36] Time of Next event
[37] Time advance

De plus, si l'on connaît la date actuelle globale de la simulation (t), il est possible de calculer le temps écoulé depuis le dernier événement traité e[38] :

$$e = t - tl$$

Enfin, la dernière variable est le temps restant avant le prochain événement planifié, notée σ :

$$\sigma = tn - t = ta(s) - e$$

La valeur de tn est systématiquement retournée par le simulateur à son coordinateur supérieur à la fin du traitement d'un événement afin de pouvoir correctement synchroniser les prochains événements à traiter entre les différents simulateurs.

[ZEIGLER 00] propose une description sous forme de pseudo code de l'algorithme du simulateur abstrait ; cette représentation est rappelée dans la Figure 23.

```
Devs-Simulator
variables
   parent //parent coordinator
   tl // time of last event
   tn // time of next event
   DEVS // Associated model with total state (s,e)
   y //current output value
when receive initialization message (i,t) at time t
   tl = t-e
   tn = tl+ta(s)
when receive internal state transition message (*,t) at time t
   if t ≠ tn then
         error: bad synchronization
   y = λ(s)
   send Ymessage (y,t) to parent
   s = δint(s)
   tl =t
   tn = tl+ta(s)
when receive input message (x,t) at time t
   if not (tl<=t<=tn) then
         error: bad synchronization
   e = t-tl
   s= δext(s,e,x)
   tl=t
   tn = tl+ta(s)
   end DEVS Simulator
```

Figure 23 - Algorithme du simulateur de modèle DEVS basique [ZEIGLER 00]

Le premier message reçu par le simulateur d'un modèle DEVS est un *Imessage(i, t)*. Ce message permet d'initialiser l'état du modèle dans un état souhaité s_{init} comme spécifié par le modélisateur. Les variables temporelles, tl et tn, sont également initialisées à : tl = t – e et tn = tl + ta(s). Pratiquement, dans le cas du démarrage de la simulation, les valeurs seront initialisées respectivement à : tl = 0 et tn = ta(s_{init}).

La réception par le simulateur d'un ***message(*, t)*, correspond à un événement interne, ce qui signifie implicitement que le simulateur n'a reçu aucun événement externe entre les dates tl et tn. La fonction de sortie du modèle associé est alors exécutée et un événement de sortie est éventuellement généré. Le simulateur émet alors un *Ymessage(y, t)* à destination de

[38] elapsed time

son parent. La fonction de transition interne du modèle est ensuite exécutée provoquant le changement d'état du modèle. Enfin, la date du dernier événement traité de la simulation tl est avancé à la date t de l'événement reçu et tn prend la valeur du nouveau tl plus la durée de vie du nouvel état. Le simulateur passe à ce moment à nouveau le relais au coordinateur supérieur et le traitement continue au cycle suivant.

La réception d'un événement externe, par l'intermédiaire d'un *Xmessage(x, t)*, par le simulateur à une date t (telle que $tl \leq t \leq tn$), provoque le calcul du temps écoulé dans l'état (noté e) en fonction de la date du message reçu et de tl. La variable e est ensuite utilisée par la fonction de transition externe, au même titre que l'événement d'entrée et l'état actuel du modèle, afin de calculer le nouvel état du modèle. Enfin l'horloge tl est avancée à t et tn prend la valeur du nouveau tl additionnée de la durée de vie du nouvel état.

Dans le cas particulier du traitement d'événements simultanés, c'est à dire si le simulateur reçoit un/ou plusieurs *Xmessage(x, t)* **et** t = = tn, une fonction de transition d'état nommée *confluente* peut être définie. Cette fonction est introduite dans Parallel DEVS [CHOW 94]. Elle traite un ensemble (« paquet ») d'événements, provoque un seul changement d'état et émet éventuellement un événement de sortie.

Un message supplémentaire nommé *Dmessage* (ou *done-message* représenté sur la Figure 22), est ajouté, dans [ZEIGLER 00], aux messages présentés précédemment. Ce message a pour fonction de transmettre explicitement les informations en retour du traitement des messages traités par les simulateurs. Il retourne notamment les nouvelles valeurs de tl et tn des simulateurs à leurs supérieurs. Il est considéré comme un message d'acquittement permettant de raporter l'éxécution de la transition au niveau du simulateur et ainsi de contrôler et vérifier la cohérence globale des messages envoyés, reçus et traités.

2.1.1.3 Coordinateurs

Les composants coordinateurs sont en charge de l'échange synchronisé des messages entre les simulateurs.

Les coordinateurs maintiennent un échéancier contenant les événements planifiés pour leurs successeurs. Les événements sont représentés par une date, une valeur et par un simulateur destinataire. Le coordinateur sélectionne l'événement de date minimale dans son échéancier et le fait parvenir par message à son successeur. Dans le cas où plusieurs événements possèdent la même date, le coordinateur fait appel à une fonction qui détermine la priorité entre les simulateurs. De plus, lors de l'initialisation, le coordinateur doit être capable de prendre en considération les événements injectés à son niveau, c'est-à-dire les messages provenant d'un port d'entrée du modèle.

```
Devs-Coordinator
variables
   parent //parent coordinator
   tl // time of last event
   tn // time of next event
   DEVN // Associated network
   event-list //ordered list of events
   d* // Selected imminent child
when receive imessage (i,t) at time t
   for each d in D do
         send i-message to child d
         sort event list
         tl = max(tld|d in D)
         tn = min(tnd|d in D)
when receive *message (*,t) at time t
   if t ≠ tn then
         error: bad synchronization
   d*= first(event-list)
   send *-message to d*
   sort event-list according to tn_d and Select
   tl =t
   tn = min {tn_d | d in D}
when receive Xmessage (x,t) at time t
   if not (tl<= t <=tn) then
         error: bad synchronization
       // consult external input coupling to get children influenced //
       by the input
   receivers={r|r in D, N in I_r, Z_N,r(x) not empty set}
   for each r in receivers
         send Xmessage (x_r, t) with input value x_r=ZN,r(x) to r
   sort event-list according to tn_d and Select
   tl=t
   tn = min {tn_d | d in D}
when receives Ymessage (yd*, t) with output y_d* from d*
   // check external coupling to see if there is an external
   // output event
   if d* in I_N and Z_d*,N(y_d*) not empty set then
         Send Ymessage with value yN = Zd*,N(yd*) to parent
   // check internal coupling to get chidren influenced by
   // output y_d* of d*
   receivers={r|r in D, d* in I_r, Z_d*,r(y_d*) not empty set}
   for each r in receivers
         send Xmessages with input value x_r=Z_d*,r(y_d*) to r
end DEVS coordinator
```

Figure 24 - Algorithme du coordinateur de modèle DEVS couplés [ZEIGLER 00]

Les coordinateurs manipulent les mêmes types de messages que les simulateurs. Le coordinateur de modèles couplés doit donc gérer les messages provenant de ses supérieurs (qu'il s'agisse d'autres coordinateurs ou du coordinateur racine) ainsi que les messages provenant de ses successeurs. A partir des messages reçus, en fonction du contenu de ces messages et des informations de couplage entre les modèles, il doit effectuer un aiguillage correct de ces messages.

Lorsque le coordinateur reçoit un *Imessage(i, t)*, il le retransmet à tous ses successeurs hiérarchiques. En retour ses successeurs font parvenir une information d'acquittement, soit sous une forme implicite, soit dans un message de retour (*Dmessage*). Cette information contient notamment les nouvelles valeurs de tl et tn. Lorsqu'il a reçu le *Dmessage* de tous ses

successeurs, il peut alors calculer la valeur de son tl = max $\{tl_d \mid d \in D\}$ et tn = min $\{tnd \mid d \in D\}$, où D est l'ensemble de ses successeurs.

Lorsque le coordinateur reçoit un **message(*, t)*, il fait appel à son échéancier pour déterminer le successeur qui est imminent et lui retransmet le message. Ce successeur traite alors ce message directement, ou en le faisant parvenir à ses propres successeurs. Il peut alors retourner éventuellement un *Ymessage(y, t)* et, dans le cas d'une utilisation explicite du message de retour, produire un message d'acquittement *Dmessage(done, t)*. Le coordinateur traite alors l'événement *Dmessage(done, t)* ou l'information de retour implicite, en actualisant sa liste d'événements et en informant à son tour ses supérieurs.

Lorsque le coordinateur reçoit un *Ymessage(y, t)* provenant de son successeur imminent actuel, il consulte les relations de couplage externe définissant les ports de sortie influencés par ce modèle pour savoir s'il doit transmettre le message à son supérieur. Il consulte ensuite les relations de couplage interne pour déterminer le successeur et les ports qui peuvent être influencés par cet événement. Dans ce cas, il transforme le *Ymessage* en *Xmessage* à destination des successeurs influencés, puis, fait parvenir un *Dmessage(done, t)* à son supérieur.

Après traitement des *Dmessage(done, t)* et *Ymessage*, le coordinateur calcule ses nouvelles valeurs de tl et tn. La variable tn est la date minimum des événements à traiter des successeurs, et la variable tl est la date du dernier événement traité à ce niveau, dont le coordinateur a reçu l'information d'acquittement de ses successeurs.

Le dernier cas correspond à la réception par le coordinateur d'un *Xmessage(x, t)* de son supérieur. Il consulte alors les relations de couplage du modèle couplé auquel il est associé et détermine tous les successeurs influencés par cet événement d'entrée. Il crée un message pour chaque influencé et le transmet aux ports des modèles successeurs. En retour, les simulateurs, destinataires directs de la structure hiérarchique de simulation ou non, traitent ces messages et font parvenir un message d'acquittement contenant leurs nouvelles valeurs de tn et tl. Enfin, à partir des valeurs reçues, le coordinateur actualise ses propres valeurs de tn et tl, et informe son supérieur du traitement de ce message.

2.1.1.4 Coordinateur racine

L'algorithme du coordinateur racine contient la boucle de simulation globale. Cette fonction assure l'émission des messages à destination du coordinateur subordonné direct, qui correspond au modèle couplé de plus haut niveau. Ce coordinateur communique à son tour avec tous les sous coordinateurs pour continuer le cycle de simulation. Les messages qui initient un cycle de simulation sont les **messages*. Le subordonné du coordinateur racine transmet cet **message* à ses successeurs et répond au coordinateur racine en lui faisant parvenir en retour la date du prochain événement à traiter nommée tn. Cette date est à son tour utilisée pour initier un nouveau cycle de simulation associé à l'émission du prochain **message*. Le cycle de simulation continue tant qu'il existe des **messages* à traiter, ou tant qu'une condition d'arrêt n'est pas satisfaite (date limite de production de certains événements, date de fin de simulation, etc.).

```
Devs-root-coordinator
   variables
         t // current simulation time
         child // direct subordinate devs-simulator or coordinator
   t=t0
   send Imessage (i, t) to child
   t=tn of its child
   loop
         send *message to child
         t = tn of its child
   until end of simulation
end devs-root-coordinator
```

Figure 25 - Algorithme du coordinateur racine du simulateur abstrait [ZEIGLER 00]

2.2 Mise à plat des modèles DEVS/G-DEVS

La structure de simulation hiérarchique classique DEVS respecte la structure hiérarchique du modèle couplé associé et conduit à utiliser autant d'échéanciers d'événements et de processeurs coordinateurs que de modèles couplés. Chaque échéancier est de taille réduite mais le nombre important de composants entraîne la gestion d'un grand nombre d'échéanciers et de messages échangés entre eux. Ce découpage est la conséquence de la description du concepteur du modèle et ne prend pas en compte la mécanique de simulation.

Par opposition, le choix d'un simulateur à échéancier unique permet de réduire le nombre de messages échangés, le problème lié à la manipulation d'un grand échéancier (insertion et classement des messages) peut être amélioré grâce à des heuristiques paramétrables en fonction des durées de vie des modèles. Nous pouvons citer en référence les travaux de [GIAMBIASI 79] pour illustrer une telle heuristique paramétrable. L'heuristique proposée consistait à définir un échéancier avec un futur proche (date limite paramétrable) et un deuxième échéancier représentant un futur plus lointain. Dans ce cas, le nombre d'échéanciers et leur gestion dépendent des paramètres du modèle simulé et des retards décrits dans celui-ci et non pas de la structure décrite par l'utilisateur. De plus, il est évident que le nombre de messages échangés dans ce genre d'approche n'est pas augmenté, les événements sont également limités dans chaque échéancier. Les mesures de performances effectuées dans [KIM 00], [WAINER 01] et [GLINSKY 02a 05] viennent étayer cette hypothèse en montrant que la structure mise à plat surpasse la version hiérarchique en durée d'exécution des modèles.

Pour les raisons énoncées ci-dessus, nous avons choisi de « mettre à plat » la structure hiérarchique de simulation ([ZEIGLER 00]) dans l'environnement que nous proposons.

2.2.1 Travaux précédents

[KIM 00] présente une méthodologie de simulation distribuée pour des modèles spécifiés dans le formalisme DEVS. La méthodologie transforme un modèle DEVS hiérarchique en un modèle non-hiérarchique. Cette transformation réduit la surcharge d'informations manipulées pendant une simulation hiérarchique séquentielle de modèles DEVS et facilite, par ailleurs, la synchronisation de la simulation distribuée, augmentant ainsi la stabilité du moteur de simulation. Pour prouver l'efficacité de la méthodologie proposée, les auteurs ont développé un en-

vironnement de simulation en Visual C++ et ont conduit une évaluation de performance de la simulation pour un système de logistique à grande échelle. Le résultat de mesure de performance met en évidence que la nouvelle méthodologie proposée fonctionne correctement et offre de meilleures performances que les approches précédentes.

[WAINER 01] et [GLINSKY 02a] ont effectué des mesures de performance de simulations mises à plat sur un ensemble de modèles, toutes les mesures concluent par de meilleures performances pour cette structure par rapport à une structure hiérarchique. Par la suite, [GLINSKY 05] a développé DEVStone, un logiciel de test consacré à l'automatisation de l'évaluation d'approches de simulations basées sur DEVS. DEVStone analyse la performance de versions successives du même moteur de simulation (e.g. suite à une mise à jour ou à la résolution d'un problème), et fournit un métrique commun pour comparer les environnements de M&S différents. Les études réalisées avec DEVStone ont notamment permis de conclure qu'en général, la technique de simulation « mise à plat » surpasse la forme hiérarchique, réduisant la surcharge d'informations traitées jusqu'a 50 % et fournit donc des temps de réponse améliorés et un meilleur pourcentage de succès dans l'exécution. Ainsi, l'utilisation de l'approche non-hiérarchique permet la simulation de plus grands modèles avec des meilleurs résultats d'exécution. Ces résultats sont une conséquence du nombre réduit de messages échangés dans le mécanisme de simulation plat.

2.2.2 Nouvelle structure de simulation

Les principes énoncés dans le § 2.2., et en particulier les travaux précédents, s'accordent tous en terme de performance de la structure « mise à plat » relativement à la structure hiérarchique. En conséquence, nous avons choisi d'utiliser, dans la simulation que nous proposons, une structure de simulation inspirée de la structure hiérarchique de simulation abstraite définie par [ZEIGLER 00] (cf. Figure 26 a) présentée dans le partie 2, mais ne comportant que deux niveaux hiérarchiques Figure 26 b). Cette structure hiérarchique est dite « compacte ».

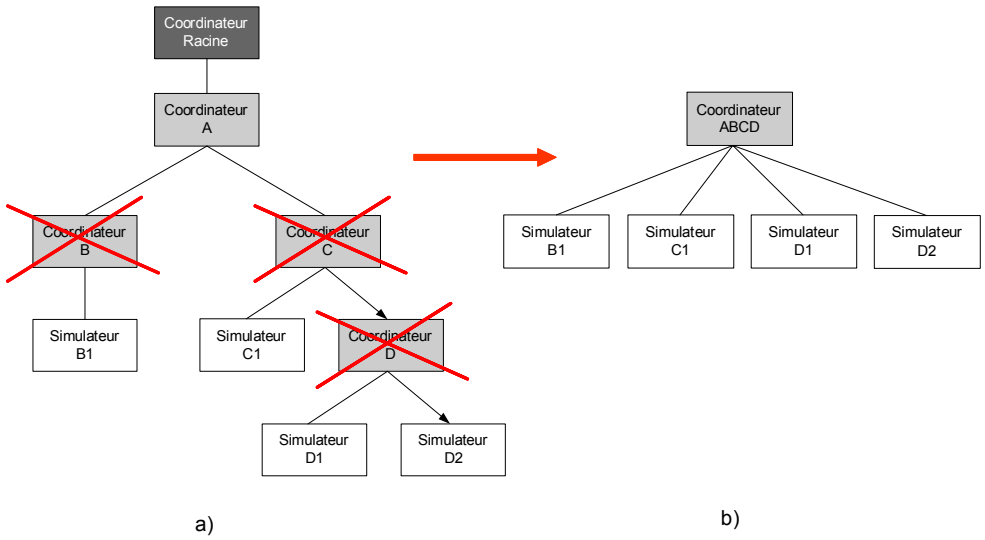


Figure 26 - Classe du modèle couplé DEVS dans LSIS_DME

A partir des travaux de [KIM 00] et dans le but de diminuer l'échange de messages entre les coordinateurs intermédiaires et les simulateurs, nous proposons donc de réduire la struc-

ture arborescente de coordinateurs intermédiaires entre le coordinateur racine et les simulateurs. Pour ce faire, nous avons choisi de ne conserver qu'un composant coordinateur auquel les composants simulateurs de modèle atomiques seront connectés en successeurs directs. La réduction de la structure de simulation est illustrée par la suppression des composants barrés sur la Figure 26 a). Cette nouvelle structure, après réduction, est présentée par la Figure 26 b).

Nous détaillons ci-dessous les fonctionnalités des différents composants de cette nouvelle structure de simulation.

2.2.2.1 Composant Coordinateur

Le composant Coordinateur (unique) regroupe les fonctions présentes dans les composants Coordinateur Racine et Coordinateur. Il devra donc gérer l'ensemble des relations de couplages EIC, EOC et IC. De plus il devra également gérer une horloge globale afin de déterminer et sélectionner le prochain événement (externe ou interne) à traiter dans l'échéancier global (Eq), en le faisant parvenir au Simulateur concerné, connecté en tant que successeur terminal direct. Nous complétons les définitions issues du simulateur abstrait par une liste d'attente (WaitList) et une liste d'acquittement (StopList) permettant de stocker les messages transférés et vérifier leurs traitements par les successeurs. Ces successeurs lui feront parvenir en retour un message d'acquittement et d'éventuels messages à insérer dans Eq.

2.2.2.2 Composants Simulateurs

Les composants Simulateurs conservent strictement la même structure que la version hiérarchique. En effet, les modifications de structure n'affectent pas ce composant qui traite les événements reçus (contenus dans des messages) en provenance de son supérieur hiérarchique. Ces événements lui permettent d'effectuer des transitions d'état du modèle atomique simulé et de retourner le résultat du traitement dans un message vers son supérieur hiérarchique.

2.2.3 Spécification de mise à plat de modèles hiérarchiques

Soit un modèle couplé N possédant la structure suivante :

$$N = (X,Y,D,\{M_d / d \in D\},EIC,EOC,IC\}$$

La définition de X et Y est identique à celle d'un modèle atomique. Les entrées et sorties sont composées de ports, chaque port peut prendre des valeurs, chaque port *p* possède son propre domaine de valeurs *v*.

$$\begin{cases} X = \{(p,v) / p \in IPorts, v \in X_p\} \\ Y = \{(p,v) / p \in OPorts, v \in Y_p\} \end{cases}$$

D est l'ensemble des noms de modèles intervenants dans le modèle couplé. M_d est un modèle DEVS atomique ou lui-même couplé. Les variables représentant les entrées et les sorties du modèle composant seront indexées par l'identifiant de ce modèle. D'où la notation suivante :

$$M_d = (X_d,Y_d,S,\delta_{ext}, \delta_{int}, \delta_{con},\lambda, ta) \text{ atomique ou}$$

$$M_d = (X_d, Y_d, D_d, \{M_{d'} / d' \in D_d\}, EIC_d, EOC_d, IC_d\} \text{ couplé}$$

Lors de la mise à plat, une analyse sera réalisée pour tous les modèles intervenants dans le modèle couplé considéré. Si le modèle M_d est atomique, il sera conservé dans la liste des modèles composants, dans le cas contraire un parcours récursif de ce modèle et de ces successeurs $M_{d'}$ permettra de remplacer dans la liste des composants de N, le modèle couplé M_d par ses sous-composants atomiques.

EIC définit les relations de couplage d'entrée du modèle, c'est-à-dire les connections entre les ports d'entrées *a* du modèle couplé *N* et les ports d'entrée *b* des modèles *d* composants le modèle couplé.

$$EIC = \{((N,a),(d,b)) / a \in IPorts, b \in IPorts_d\}$$

Autrement dit, c'est l'ensemble des ports d'entrée *IPorts* du modèle couplé associés aux ports d'entrée $IPorts_d$ des modèles D composants le modèle couplé. Lors de la mise à plat une analyse de chaque port *b* devra être effectuée afin de déterminer la nature du modèle *d* possesseur de ce port. Si *d* est un modèle couplé, il devra être déterminé à quel port d'entrée *b'* d'un sous-modèle atomique m_a composant (direct ou indirect) de *d* est connecté *b*. Cette opération pourra être assurée par un parcours récursif des EIC des sous-composants jusqu'a atteindre le (ou les) modèle(s) cible(s) atomique(s) m_a. Le (ou les) couple(s) (m_a, b') remplacera(ont) la relation de couplage *(d, b)* dans le EIC de N.

EOC définit les relations de couplage de sortie du modèle, c'est-à-dire les connections entre les ports de sortie *a* du modèle couplé *N* et les ports de sortie *b* des modèles *d* composants *N*.

$$EOC = \{((d,b),(N,a)) / b \in OPorts_d, a \in OPorts\}$$

De façon similaire aux relations EIC, lors de la mise à plat des relations EOC, une analyse de chaque port *b* devra être effectuée afin de déterminer la nature du modèle *d* possesseur de ce port. Si *d* est un modèle couplé, il devra être déterminé à quel port de sortie *b'* d'un sous-modèle atomique m_a composant (direct ou indirect) de *d* est connecté *b*. Cette opération pourra être assurée par un parcours récursif des EOC des sous composants jusqu'à atteindre le (ou les) modèle(s) source(s) atomique(s) m_a. Le (ou les) couple(s) (m_a, b') remplacera(ont) la relation de couplage *(d, b)* dans le EOC de N.

A l'intérieur du modèle couplé, les sorties d'un modèle peuvent être couplées aux entrées des autres modèles. Une sortie d'un modèle ne peut pas être couplée à l'une de ses entrées.

$$IC = \{((i,a),(j,b)) / i, j \in D, i \neq j, a \in OPorts_i, b \in IPorts_j\}$$

La mise à plat des relations IC emploie la méthode de recherche des modèles atomiques reliés aux ports source *a,* présentée dans la mise à plat des relations EIC et la méthode définie pour les relations EOC pour les ports cible *b*.

Nous présenterons dans le chapitre suivant la spécification de l'algorithme de mise à plat des modèles qui a été retenue pour le développement du moteur de simulation DEVS. Cet

algorithme de mise à plat sera décomposé en deux fonctions : la première considérant la mise à plat des modèles hiérarchiques, la deuxième permettant d'actualiser les relations de couplages en fonction des nouveaux modèles pris en compte dans la structure mise à plat.

3 Simulation distribuée de modèles G-DEVS

Le second objectif de ce chapitre est de proposer une structure de simulation distribuée pour des modèles G-DEVS.

Nous donnons sur la Figure 27 un schéma conceptuel illustrant les différentes entités intervenant dans cette structure de simulation distribuée après mise à plat. Nous avons choisi, pour atteindre cet objectif, de réutiliser localement les structures de simulation non hiérarchiques, spécifiés dans le § 2.2, afin d'exploiter les atouts, énoncés précédemment, de cette structure de simulation. En contre partie, les composants de ces structures s'exécutent, à présent, dans un contexte distribué ; elles doivent donc être en mesure de communiquer entre elles pour reconstituer un comportement global. Les liaisons entre les groupes Ordinateur 2 & 3 schématisés par des traits pointillés sur la Figure 27 illustrent la communication nécessaire dans le cadre d'une simulation globale distribuée. Nous déduisons de ce schéma conceptuel la nécessité d'ajouter des fonctions aux composants de simulation afin de gérer et de synchroniser les messages échangés entre les différents composants distribués.

3.1 Composants de simulation

3.1.1 Coordinateur local

Nous allons réutiliser en particulier le composant **Coordinateur**. L'essentiel du comportement de ce composant, caractérisé par la manipulation d'un échéancier (Eq) contenant les *Xmessages* insérés à son niveau et les ***messages* déduits des durées de vie de ses successeurs, est conservé. De même, il conserve les relations de couplage avec ses successeurs dans les listes (EOCList, ICList, EICList). Enfin, ce composant gère toujours une horloge (devenue locale dans le nouveau contexte).

En résumé, le groupe LCS[39] doit continuer à assumer la gestion de la simulation locale d'un modèle couplé de façon autonome. Mais ce groupe étant intégré à une simulation distribuée, il doit, de surcroît, s'assurer de la gestion cohérente des messages provenant des autres coordinateurs distribués. Pour cela, les fonctionnalités Coordinateur, devenu **Coordinateur « Local »** (LC) (cf. groupes Ordinateur 2&3 Figure 27), devront être étendues. En particulier, lors de la sélection du prochain message chronologique de son échéancier par rapport à sa date actuelle, et avant la transmission de ce message au simulateur successeur concerné, le LC devra s'assurer de ne pas recevoir de message, de date inférieure ou égale, provenant d'un autre composant distribué. Il devra également assurer la diffusion de ses événements de sortie vers les autres composants distribués.

[39] Local Coordinator - Simulator(s)

3.1.2 Simulateur

Les composants **Simulateurs** n'étant pas en lien direct avec la partie distribuée de la structure de simulation, ils conservent les fonctions décrites dans le simulateur conceptuel [ZEIGLER 00].

3.1.3 Coordinateur Racine Distribué

Pour assurer la communication et la synchronisation globale de la simulation, nous avons opté pour l'ajout d'un composant permettant de centraliser l'échange des informations. Cette solution permet de faciliter la synchronisation grâce à une vision globale de l'échange de messages. Ce composant est nommé **Coordinateur Racine Distribué**, il sera par la suite noté DRC[40]. Le composant DRC pourra être exécuté sur un ordinateur différent (groupe Ordinateur 1 sur la Figure 27), des ordinateurs hébergeant des groupes LCS.

Ce coordinateur distribué a pour fonction l'échange de messages entre les composants distribués LCS ; il doit donc synchroniser les envois et les réceptions de messages en respectant la causalité des événements. Sur le même principe que les coordinateurs locaux, il utilise un échéancier (Eq) contenant les messages échangés dans la simulation globale, une liste d'attente (WaitList), et une liste d'acquittement (StopList). Il comporte également une liste (Lq) contenant les valeurs de Lookahead des différents successeurs (dont l'utilité est explicitée dans la section § 3.3.3), une liste des coordinateurs en attente de réponse d'une demande d'événements (Wq), et un ensemble de tables contenant les relations de couplage entre les modèles couplés distants (EICList, EOCList, ICList).

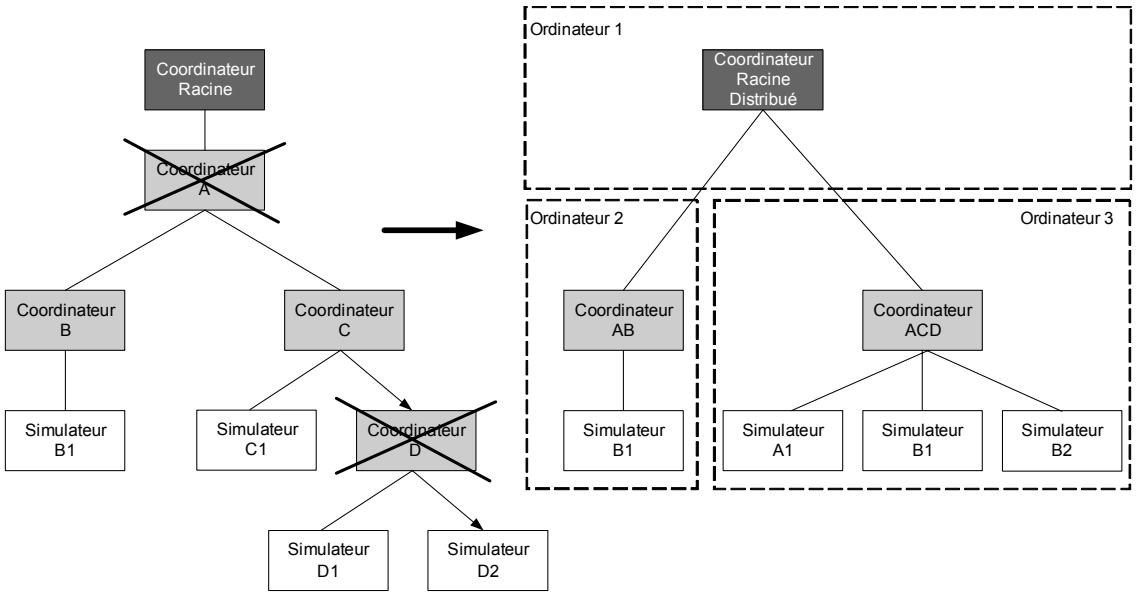


Figure 27 - Mise à plat et distribution de la structure de simulation hiérarchique

[40] Distributed Root coordinator

3.2 Algorithmes de traitement des événements

Dans le chapitre 1 § 3.3.3, nous rappelons les deux principaux mécanismes de synchronisation (pessimiste et optimiste) des simulations distribuées déterministes.
[NUTARO 03] et [GLINSKY 06] ont proposés deux environnements de simulation DEVS basés sur l'approche optimiste TimeWarp [JEFFERSON 85]. Ces environnements mettent en œuvre le mécanisme de Rollback, la mémorisation des états et des messages échangés de l'approche optimiste. Dans ses travaux de thèse, [NUTARO 03] propose un environnement pour la modélisation de systèmes hybrides et décrit leurs simulations distribuées par l'approche optimiste. Parallèlement, Les travaux de [GLINSKY 02b] ont abouti à l'élaboration d'un environnement DEVS distribué avec un simulateur simplifié utilisant également l'approche optimiste. De plus, [GLINSKY 06] présentent des performances de simulation, pour leur environnement, surpassant l'approche conservative.

Nous nous heurtons cependant à deux limitations vis-à-vis de l'utilisation du mécanisme de synchronisation optimiste.

D'un coté, la littérature mentionne un échange important de messages provoqué par les situations de RollBack de cette approche (notamment, dans [ZEIGLER 97] et [FUJIMOTO 98]) et donc une surcharge en temps d'exécution relativement à la version pessimiste. Ceci, en particulier dans le cas des simulations faisant intervenir un nombre important de messages, dont les composants distribués sont nombreux et/ou les durées absolues de simulation (cf. chapitre 1 § 2.3.4.2) sont très différentes entre les différents processeurs. De ce fait, les mesures de performances de [GLINSKY 06] sont à relativiser à l'étude de modèles particuliers dans un contexte de simulation donné.

D'un autre coté, l'objectif final de l'environnement de simulation distribué, introduit dans cette thèse, est de s'intégrer dans une application Workflow. Cette application Workflow doit permettre de faire intervenir des applications hétérogènes (i.e. autres que l'environnement de M&S G-DEVS distribué) dont certaines ne fournissent pas leurs comportements internes. Il est donc difficile, dans ce cas, de mettre en œuvre un mécanisme de mémorisation des états pour ces applications.

Pour assurer la synchronisation globale des simulateurs, nous avons donc choisi, pour les raisons énoncées précédemment, d'élaborer des algorithmes de synchronisation conservative basée sur les travaux de [BRYANT 77] et [CHANDY 81]. A ce titre, nous utilisons, dans la spécification des algorithmes, les notions de *LookaheadMessage* et de *NullMessage* [SAMADI 84] associées à la synchronisation conservative dont nous avons présenté la définition au chapitre 1. Nous présenterons plus en détails l'utilisation de ces messages dans les algorithmes des composants ci après.

Chaque composant (cf. § 3.1) comporte donc des fonctions pour traiter localement les messages, ainsi que des fonctions, mettant en œuvre les mécanismes de synchronisation conservative, pour interagir avec les autres composants distribués. Nous présentons ci-après les modèles comportementaux et les principales fonctions en pseudo-code des trois types de composants.

3.2.1 Coordinateur Local

Nous présentons Figure 28, le modèle comportemental du coordinateur local que nous allons commenter dans les paragraphes suivants.

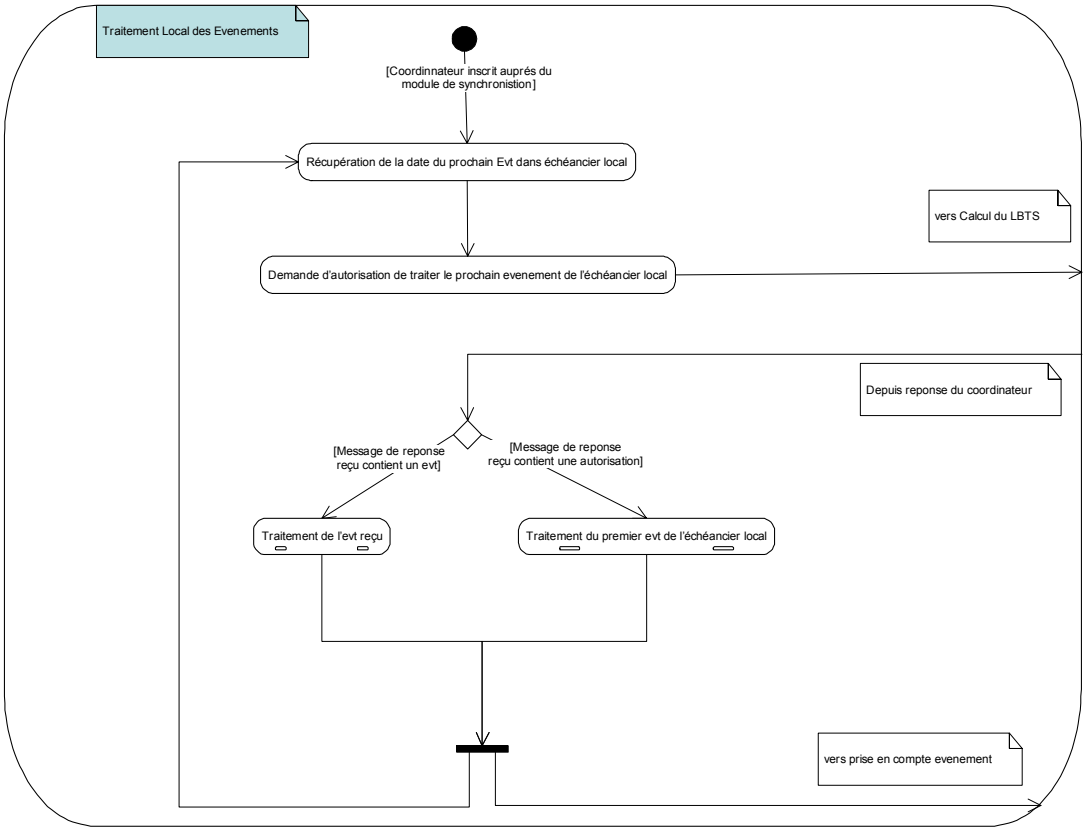


Figure 28 - Modèle comportemental du coordinateur local

Lors de son initialisation, le coordinateur local doit, dans un premier temps, s'enregistrer au niveau du coordinateur racine en lui précisant ses relations de couplage. Il lui envoie donc un message contenant son nom et les relations de couplage du modèle qu'il simule, avec les autres modèles distribués. Ces relations peuvent être représentées et décomposées en EOC, EIC et IC comme dans un modèle DEVS couplé classique.

Après avoir été enregistré par le DRC, le coordinateur local utilise une boucle de traitement des événements de son échéancier dans laquelle il sélectionne le (ou les) prochain(s) événement(s) de son échéancier. Cette boucle est présentée par la Figure 29.

```
Event-Loop:
   Ask for Join-Federation-Request (Coordinator, EIC, EOC, IC)
   do      ('*' or 'x', Simulator, t, - , - ) is first in Eq // t = null if Eq is empty
               Ask for Next-Event-Request (Coordinator, InfluencedPort, TLocal, t)  //  to
               Conservative-Distributed-Root-Coordinator pending DRC answer
   While true
   Endwhile
End Event-Loop
```

Figure 29 - Algorithme boucle de traitement du coordinateur local

Ce coordinateur local faisant partie d'une simulation distribuée, il est régi par un mécanisme de synchronisation pessimiste, qui doit assurer le respect du principe de causalité. En d'autres termes, le coordinateur doit être certain de ne pas recevoir, dans la suite de

l'exécution de la simulation, un événement ayant une date inférieure à la date de l'événement qu'il a sélectionné. Pour assurer le respect de cette contrainte, l'algorithme du coordinateur local possède une fonction d'appel au coordinateur racine distribué nommée *NextEventRequest()*. Cette fonction lui permet d'informer le DRC de son souhait de traiter un événement local de date donnée et de s'assurer ainsi de la possibilité de le traiter sans omettre de prendre en compte les influences d'autres simulateurs répartis. Le coordinateur passe alors dans une phase d'attente de réponse de la part du DRC.

La réponse de l'appel au DRC se présente sous plusieurs formes. Il peut s'agir d'une autorisation de traitement de l'événement local que le LC avait sélectionné dans son échéancier, d'un *Xmessage* de date inférieure ou égale à la date de l'événement local sélectionné. Nous présenterons en détail la détermination et la constitution de la réponse du Coordinateur Racine dans § 3.2.3. Nous allons à présent détailler le traitement par le LCS des différents types de réponses données par le DRC.

La Figure 30 présente le cas où la réponse du DRC est une autorisation. Le DRC a déterminé ici que le LCS ne sera pas influencé jusqu'à la date du premier message de sa liste Eq. Ce premier message de l'échéancier est donc transmis au simulateur successeur imminent.

```
When receive Autorisation (Coordinator, TNextLocal)
   add Autorisation (Coordinator, TNextLocal) to WaitList
   if (first of Eq t = TNextLocal)
   then    send first of Eq message ('*', Simulator, t, - , - ) to concerned simulator
           Add first of Eq message ('*', Simulator, t, - , - ) to WaitList & mark it
           "send"
           remove first of queue Eq
           tl = t
           tn = t first Event of the EventList
   else error: bad Synchronisation
```

Figure 30 - Algorithme réception Autorisation du coordinateur local

Nous illustrons dans la Figure 31 (ci-dessous) le cas où la réponse du DRC contient un *Xmessage*. Ce message est daté antérieurement au premier message de l'échéancier, il sera donc directement transmis au simulateur successeur imminent. Notons que le coordinateur peut également recevoir une variante du *Xmessage* nommée *Nullmessage*, qu'il transmet de façon similaire au successeur concerné.

```
When receive Xmessage ('x', Coordinator, t, InfluencedPort, Value)
   if (tl<= t <= tn)
   then
          Get-External-Influenced-By (Coordinator, InfluencedPort)
          // consult external input coupling to get children influenced by the input
          for each Simulator influenced by input do
                 add Xmessage ('x', Simulator, t, InfluencedPort, Value) to WaitList
                 // according to priority definition
          send First Xmessage ('x', Simulator, t, InfluencedPort, Value) of Wait List &
          mark it "send"
          tl = t
          tn = t first Event of the EventList
   else error : bad Synchronization
```

Figure 31 - Algorithme réception Xmessage du coordinateur local

En résumé, le coordinateur local sélectionne l'événement extrait de son échéancier où l'événement reçu. Dans le cas d'événements simultanés, s'ils sont dirigés vers différents simulateurs, le coordinateur local devra définir des règles de priorité entre les simulateurs ; en revanche s'ils sont dirigés vers un même simulateur, il lui fera parvenir un « paquet » d'événements simultanés comme cela sera défini dans le chapitre suivant.

```
When receive Dmessage ('d', Simulator, t, - , Value)
// inform when an Event as been processed by a hierarchical children
  add ('d', Simulator, t, - , Value) in StopList

  if (WaitList != StopList)
         then    Mark "send" first "non send" message of WaitList
                 send this message of WaitList
         else    remove these messages from WaitList and StopList
                 send Dmessage to Parent
  remove ('*', Simulator, next_internal_EventTime, - , - ) from Eq
  // last internal event associated to this Simulator
  if (value != null)
         then    extract next internal event ('*', Simulator, Value, - , - )
         // value given by Dmessage
                 add ('*', Simulator, Value, - , - ) in Eq

  tl = t
  tn = t first Event of the EventList
```

Figure 32 - Algorithme réception Dmessage du coordinateur local

Après avoir fait parvenir le message à son successeur, le coordinateur local reçoit un message d'acquittement (*Dmessage*) du successeur (voir Figure 32), qu'il insère dans sa liste d'acquittement, et compare sa liste d'attente à sa liste d'acquittement (StopList). Si le contenu des deux listes est différent, alors il existe des messages en attente d'émission, qu'il envoie aux successeurs. Si le contenu est identique, il vide le contenu des deux listes. Il extrait ensuite du *Dmessage* le prochain événement interne prévu pour le simulateur émetteur, l'ordonne dans son échéancier et supprime l'éventuel ancien événement interne planifié en EventList pour ce simulateur. Enfin il informe son supérieur de son acquittement.

```
When receive Lookahead message ('l', Simulator, t, OutputPort, Lookahead Value)
   add Lookahead (Simulator, t, OutputPort, Lookahead Value) in Lq
   Compute-Lookahead (OutputPort)       // Lookahead = min of Lookahead Value for a Output
                                        // Port of a local model according to coupling
   send Lookahead ('l', Coordinator, t, OutputPort, Lookahead Value) to DEVS Conserva-
                  tive-Distributed-Root-Coordinator

   Compute-Lookahead (OutputPort)
   Lookahead = min of (influencer Lookaheads for an Output port in Lq)
   // influencer are defined according to coupling store in EOC List)
```

Figure 33 - Algorithmes de gestion du Lookahead du coordinateur local

L'algorithme de la Figure 33 présente la réception, simultanée à celle du *Dmessage*, par le LC de la nouvelle valeur du Lookahead du simulateur. Il calcule alors la nouvelle valeur de Lookahead pour le couple local modèle/port à faire parvenir au DRC.

```
When receive Ymessage ('y', Simulator, t, Simulator_Port, Value)
   Get-Internal-Influenced-By (Simulator, Simulator_Port)
   // check for internal coupling to get children influenced by output
   for each children influenced by output
               do Add Xmessage ('x', Simulator_Influenced, t, Port_influenced, Value) in
                  WaitList
                  // according to priority definition
   Send first Xmessage ('x', Simulator_Influenced, t, Port_influenced, Value) of WaitList
   & mark it "send"

   Get-External-Influenced-By (Simulator, Simulator_Port)
   // check for external coupling to see if there is an external output event
   if (existing influenced model)
          then    send Ymessage ('y', Coordinator, t, Coordinator_Port, Value) to Parent
   tl = t
   tn = t first Event of the EventList

   B = Get-Internal-Influenced-By (A) // According to the couple ((CoordA, PortA) (CoordB,
   PortB)) in the IC List, the Output PortA influences the Input PortB

   A = Get-External-Influenced-By (C) // According to the couple ((CoordC, PortC) (CoordA,
   PortA)) in the EIC List, A is an internal model of C and the Intput PortC influences
   the Input PortA
```

Figure 34 - Algorithme réception Ymessage du coordinateur local

Lorsqu'un coordinateur local reçoit de la part d'un successeur un événement de sortie (sous la forme d'un *Ymessage* Figure 34), il vérifie, en fonction des relations de couplage, si un de ses successeurs est influencé par cet événement ; dans ce cas, il le transforme en événement d'entrée (*Xmessage*), le stocke en liste d'attente et le fait parvenir au (ou aux) successeur(s) concerné(s). Si le modèle local possède un port de sortie, un *Ymessage* est généré à destination du coordinateur racine.

Enfin, après réception et traitement des messages de retour provenant des simulateurs successeurs, le coordinateur initie un nouveau cycle de simulation.

3.2.2 Simulateur

Le simulateur de la structure LCS distribuée calcule, identiquement à la version non distribuée, le changement d'état et éventuellement l'événement de sortie conséquent à la réception d'un événement, selon les fonctions du simulateur basique présentées par la Figure 23. Il possède cependant une fonction de traitement supplémentaire de traitement de *Nullmessage*, cette fonction ne provoque pas de changement d'état, elle permet uniquement de recalculer une éventuelle nouvelle valeur de lookahead.

La Figure 35 présente le modèle comportemental du simulateur.

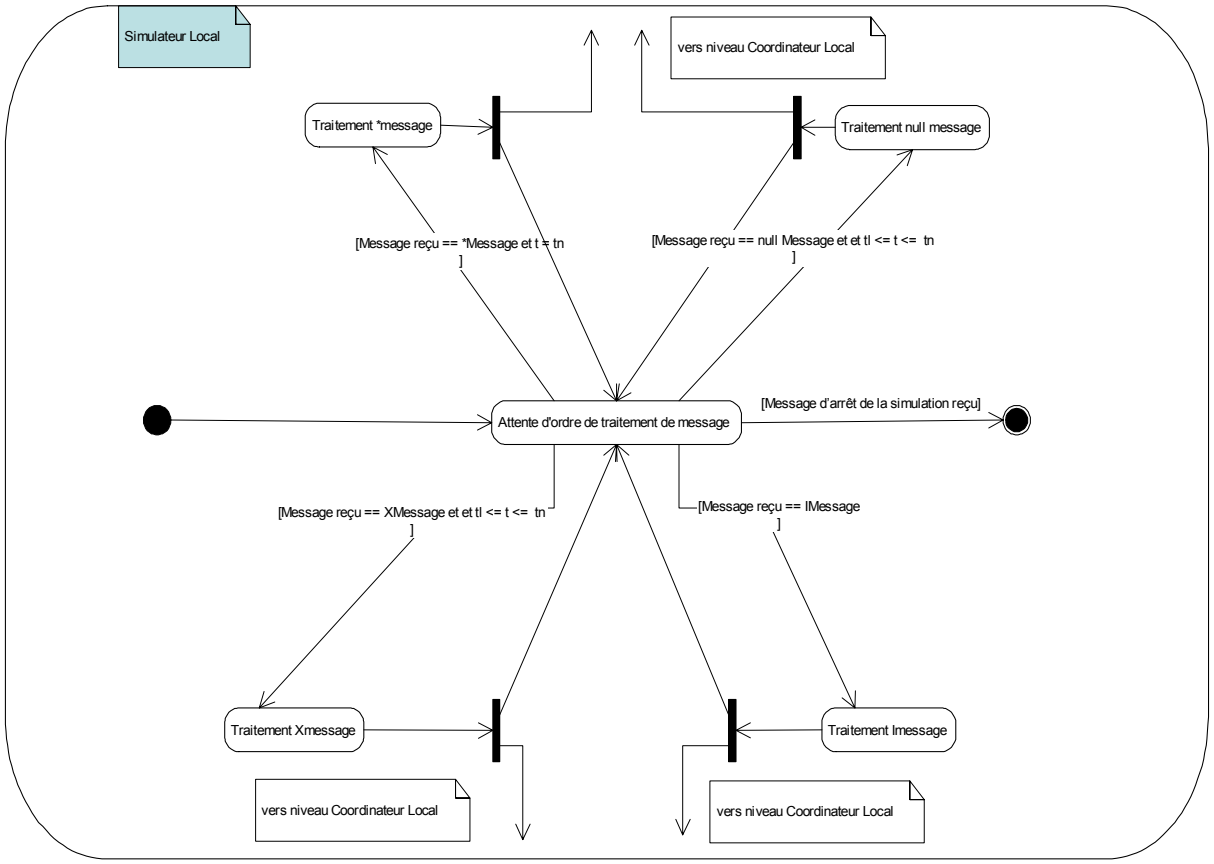


Figure 35 - Modèle comportemental du niveau simulateur

Dans le but de fournir des informations supplémentaires à la simulation distribuée, nous avons complété le code du simulateur par une fonction déterminant en outre une nouvelle valeur de Lookahead dépendante du modèle. Le calcul de cette valeur est défini par la relation suivante (1). Ce calcul est effectué au cours du traitement des messages.

Lookahead = min (DDV_s | s ∈ S) Avec S : ensemble des phases du modèle atomique. (1)
Cette information est jointe au message de retour à destination du coordinateur local. La Figure 36 (ci-dessous) présente un calcul du Lookahead de façon informelle.

```
compute Lookahead
       if (actual state has only internal transition)
       then    Lookahead = tn
       else    if (actual state has (only external transition)
                      OR (external & internal transition))
               then Lookahead = min tn of next state reachable with only external transition
```

Figure 36 - Calcul du Lookahead au niveau simulateur

Nous reviendrons en détails dans la suite de ce chapitre, sur le calcul du Lookahead. En effet, l'algorithme Figure 36 fait apparaître que cette valeur peut être calculée en fonction du comportement dynamique du modèle. Pour ces raisons, nous nous efforcerons d'améliorer le calcul du Lookahead des modèles DEVS en prenant en considération l'évolution des variables du modèle au cours de la simulation.

3.2.3 Coordinateur Racine Distribué

Nous présentons ici le comportement (Figure 38) et les fonctions (Figure 37) du composant DRC. Initialement le coordinateur racine est en attente des inscriptions initiales des LCS (fonction décrite Figure 29). Il possède une fonction lui permettant de compléter ses listes conservant le couplage entre les différents LCS.

Après avoir reçu les informations d'enregistrement de tous les LCS, le coordinateur racine distribué passe dans un état d'attente des demandes d'autorisation `Next-Event-Request()` de traitement des messages locaux provenant des LCS comme défini dans l'algorithme de la Figure 37.

```
When receive for Next-Event-Request (Children_Coordinator, InfluencedPort, TLocal, TNextLocal)
        Get-Internal-Influencer-Of (Children_Coordinator, InfluencedPort)
        Compute-LBTS (Children_Coordinator, InfluencedPort) // using the Lookahead min of the
                                    // influences of the model that is simulated by Chil-
                                dren_Coordinator
        if (there is a Message for Children_Coordinator in Eq with timestamp TNextGlobal < LBTS
                        & TNextGlobal < TNextLocal)
        then    Add in WaitList (Message) & mark it "send"
                send (Message) // as an answer to the request
                else    if (TNextLocal < LBTS)
                        then    Add in WaitList Autorisation (TNextLocal) & mark it "send"
                                send Autorisation (TNextLocal) // as an answer to the request
                        else    if ((TNextLocal >= LBTS) or (TNextLocal == null)) & (TLocal =! LBTS)
                                then Add in WaitList nullmessage("null", coordinator, LBTS,
                                                                InputPort,-) & mark it "send
                                                    send nullmessage("null", coordinator, LBTS, InputPort,-)"
                                                // as an answer to the request
                        else    if (TLocal == LBTS)
                                        then add (Children_Coordinator, TLocal, TNextLocal) in Wq
```

Figure 37 - Algorithme NextEventRequest du coordinateur racine

Lorsque le DRC reçoit une demande en provenance d'un coordinateur local, dont la date actuelle est `TLocal`, concernant une autorisation de traitement d'un événement local de date `TNextLocal`, il doit déterminer si ce coordinateur local demandeur ne recevra pas d'influence avant `TNextLocal`.

L'algorithme du coordinateur racine définit donc une date minimale (LBTS défini § 3.4.4.9 du chapitre 1), à partir des valeurs de Lookahead des modèles influenceurs, jusqu'à laquelle le LCS influencé ne recevra pas d'événement. En fonction de ces données, le coordinateur racine émet une réponse de type : Autorisation, *Xmessage* ou *Nullmessage*. Le schéma Figure 38 comporte le positionnement temporel des différentes dates permettant de déterminer la réponse du DRC à une demande d'un LCS.

Dans le cas où `TNextLocal` est inférieur au LBTS et que le DRC ne comporte pas de message ou un message de date supérieure à celle de `TNextLocal` à destination du LCS, il donne au LCS une autorisation de traiter son message local.

Dans le cas où `TNextLocal` est inférieur au LBTS et que le DRC comporte un message à destination du LCS de date inférieure au LBTS, il délivre ce message de date `TNextGlobal` au LCS.

Dans le cas où `TNextLocal` est supérieur au LBTS et que le DRC ne comporte pas de message à destination du LCS de date inférieure au LBTS, il délivre un *Nullmessage* de date LBTS au LCS. Ce message permet au LCS d'avancer son horloge et ainsi d'éventuellement débloquer d'autres LCS qu'il influence.

Dans tous les autres cas, la demande du LCS est suspendue en attente de réponse du DRC, qui attend la modification des informations des influenceurs du LCS considéré.

En retour des autorisations et messages émis vers les LCS, le coordinateur racine reçoit un message d'acquittement, il l'insère dans sa liste de message d'acquittement, qu'il compare à sa liste d'attente et supprime alors les messages identiques dans les deux listes. Il extrait également le prochain événement interne (**message*) de son successeur et met à jour son échéancier. Enfin, il récupère la nouvelle valeur de Lookahead de ce successeur qu'il utilise pour déterminer un nouveau LBTS pour les influencés du successeur émetteur de l'acquittement. Lorsque le coordinateur racine reçoit un événement de sortie sous la forme d'un *Ymessage*, il le transforme en événement d'entrée (*Xmessage*) en fonction des relations de couplage et le stocke alors dans son échéancier.

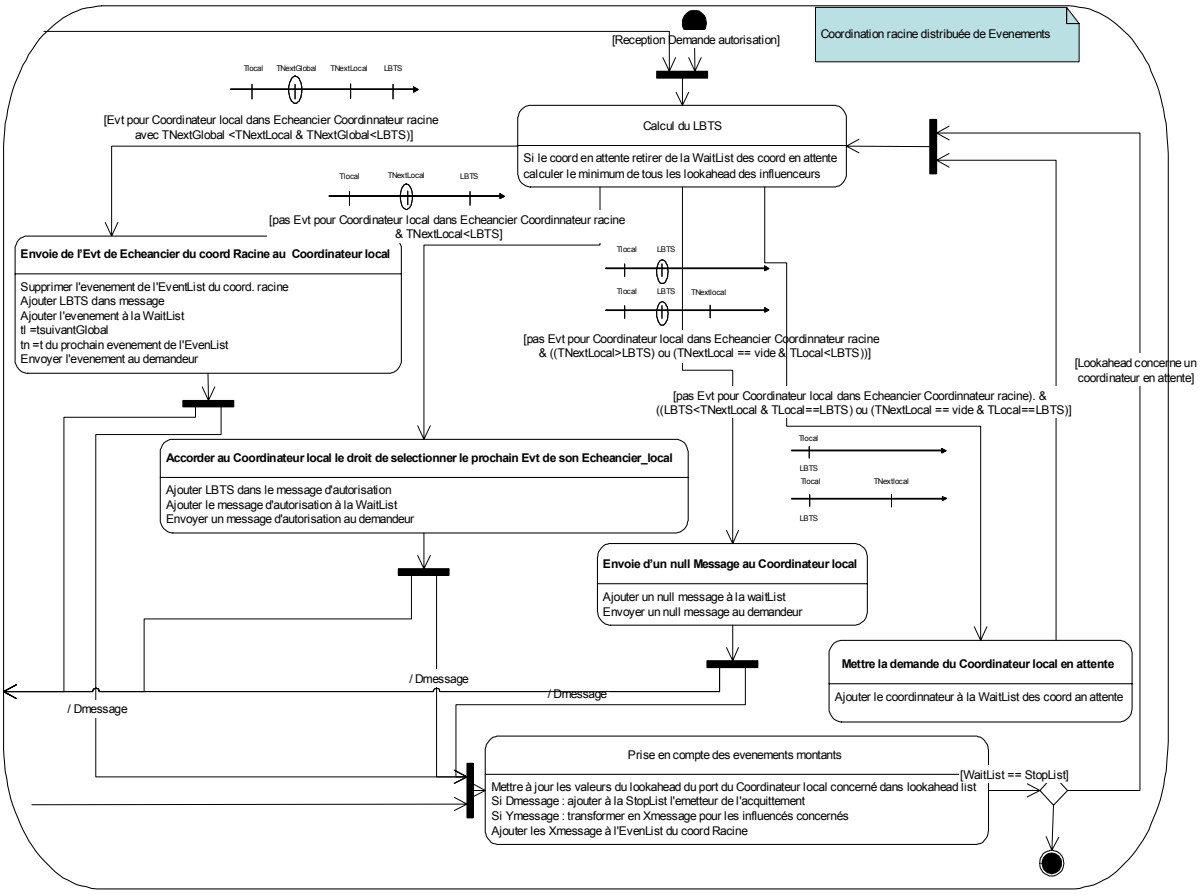


Figure 38 - Modèle comportemental du coordinateur racine

3.3 Définition de modèles couplés distribués conformes à HLA

La spécification de simulation distribuée HLA paraît particulièrement adaptée pour la mise en œuvre de la structure de simulation distribuée présentée précédemment. En effet, cette norme permet la réutilisation et l'interopérabilité de composants de simulation sans nécessiter de les recoder. De plus, l'implémentation de la spécification d'interface (RTI) propose des services qui peuvent inscrire ceux définis dans le composant Coordinateur Racine Distribué. Enfin, les ensembles Coordinateurs Locaux et Simulateurs peuvent être considérés comme des fédérés HLA interagissant entre eux pour réaliser la simulation d'un modèle structurel global (Fédération).

Nous proposons, dans la section suivante, un rappel des précédentes intégrations de modèles DEVS dans une structure de simulation distribué respectant la norme HLA.

3.3.1 Rappel d'Intégrations DEVS/HLA

Le premier algorithme d'intégration a été présenté par [ZEIGLER 99]. Pour garantir la synchronisation globale des Coordinateurs Locaux, cette solution exploite un mécanisme d'algorithmes conservateurs basé sur les travaux de [BRYANT 77] et [CHANDY 81] proposés dans la mise en œuvre de la spécification HLA. Cette méthode présente l'avantage de réduire l'échange de messages par rapport à l'approche optimiste où les situations de Repositionnement (*Rollback*) génèrent un nombre important de messages échangés [JEFFERSON 85].

La lacune de cette première solution résulte du traitement des événements simultanés par les fédérés qui sont à la fois influencés et influenceurs. La fonction de rappel du RTI *TimeAdvanceGrant*(T)† accorde à un fédéré influenceur (ou imminent[41]) le droit de traiter un événement de date T. Un fédéré non imminent doit attendre que tous les fédérés imminents aient émis leurs interactions de date T' ≥ T pour recevoir *TimeAdvanceGrant*(T)†. Dans le cas d'une influence cyclique entre fédérés influencés et influenceurs, il est impossible de déterminer à quel fédéré accorder *TimeAdvanceGrant*()† en premier.

En utilisant le travail de [ZEIGLER 99] comme référence, les travaux de [LAKE 00] ont présenté deux approches d'intégration des modèles DEVS dans HLA. Ils ont tout d'abord précisé une distinction entre « Lookahead DEVS » (cf. relation (2)) et « Lookahead HLA ».

$$\text{Lookahead DEVS} = \text{Min } ta(s) \,/\, s \in S \text{ avec } S \text{ ensemble des états du modèle} \quad (2)$$

Le « Lookahead HLA » est, comme précisé au premier chapitre, un retard minimal du temps simulé local du fédéré et jusqu'auquel le fédéré assure de ne pas émettre d'événement. Les auteurs ont présenté ici une mise en oeuvre directe de DEVS dans HLA qui résout le problème rencontré par [ZEIGLER 99]. Pour cela, ils ont utilisé le service HLA *NERA*(T)[42] défini par [FUJIMOTO 97]. Si le *TimeAdvanceGrant*(T)† est une réponse à *NERA*(T), le fédéré

[41] Un fédéré influenceur peut publier des informations datées

[42] NextEventRequestAvailable(T)

peut (si son Lookahead est nul) émettre un événement daté également de T, par contre, le fédéré n'est pas sûr d'avoir reçu tous les événements daté T résultant de la fédération. Cette solution utilise ici un Lookahead HLA nul pour chaque fédéré.

La deuxième solution de [LAKE 00] propose de diffuser les événements à tous les fédérés dans un message et de donner à tous les fédérés une connaissance globale des relations de couplage. Les entités locales décident d'un traitement de message quant à leur historique des messages reçus et à leur connaissance des relations de couplage. Dans ce cas, il est possible de définir un Lookahead HLA constant et non nul. Mais le problème réside dans le transfert de responsabilités du RTI vers les entités de simulation. Nous considérons que ce déplacement réduit les responsabilités et donc l'utilité du RTI.

3.3.2 Création d'une fédération suivant le FEDEP

Nous proposons une nouvelle intégration des modèles couplés DEVS ou G-DEVS dans une structure de simulation distribuée compatible HLA. Notre solution se base sur celle de [LAKE 00], cependant elle propose un cycle de développement normalisé en se conformant au cycle de développement de fédérés et de fédérations proposé par HLA nommé FEDEP[43] [IEEE1516.3 03].

Le FEDEP n'est pas une obligation de HLA mais il préconise six étapes dans le processus de création d'une fédération qui formalise et aide à réutiliser les développements.

Le premier point consiste à définir les objectifs de la fédération. Dans notre cas, les fédérations créées permettent de définir des modèles couplés DEVS/G-DEVS distribués.

Nous générons ensuite un modèle conceptuel comme cela est préconisé dans le point numéro deux du FEDEP. Ce modèle contient le modèle G-DEVS représenté par des entités et des actions représentant les événements externes échangés entre les coordinateurs G-DEVS. Cette seconde étape décrit donc comment vont s'échanger les messages représentant les événements entre les coordinateurs locaux en respectant la causalité globale.

Dans les références [ZEIGLER 98a] [ZEIGLER 98b] [ZEIGLER 99], les auteurs ont présenté une première intégration des Coordinateurs DEVS dans une architecture compatible HLA. Ces références intègrent les modèles couplés DEVS locaux dans des fédérés HLA, dont les coordinateurs de plus haut niveau ont pour responsabilité de gérer la simulation locale, mais aussi de communiquer avec un fédéré nommé « Manager de Temps ». Le troisième pas FEDEP demande la description des fédérés participant à la fédération HLA. Nous avons choisi à ce stade d'intégrer chaque processeur LCS introduit au § 3.1.1 dans un fédéré HLA conformément à [ZEIGLER 99], mais nous n'utiliserons pas le fédéré « Manager de Temps ». Nous avons choisi d'intégrer directement les fonctionnalités du composant DRC dans le RTI. En effet, la spécification d'interface (RTI) propose des services qui incluent ceux définis dans le DRC. Ainsi, le modèle « global distribué » (c'est-à-dire la fédération) est constitué de fédérés LCS communiquant réciproquement via la RTI (cf. Figure 39).

[43] HLA Federation Development and Execution Process

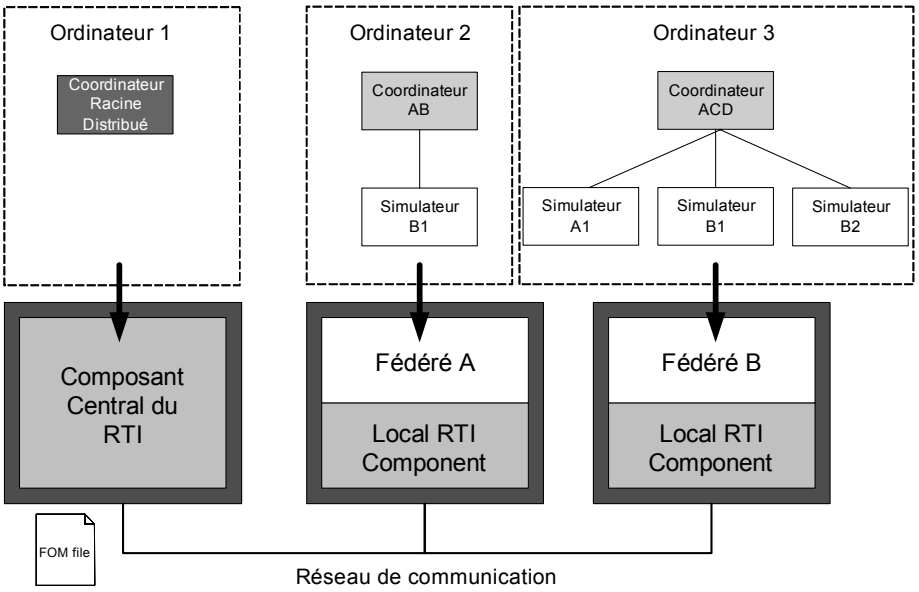


Figure 39 - Intégration du simulateur G-DEVS distribué avec le RTI

Dans le quatrième pas FEDEP, nous intégrons les relations de couplage des modèles G-DEVS dans des interactions HLA comme dans [ZEIGLER 99]. Chaque modèle G-DEVS couplé et son LCS associé possèdent donc un SOM définissant les informations qu'ils sont capables de fournir dans une simulation distribuée respectant HLA. Les données communes des SOM des différents fédérés permettent de produire le FOM de la fédération DEVS-G-DEVS/HLA (cf. tables Figure 40, Figure 41). Plus précisément, un modèle G-DEVS possédant un port de sortie « publie » sur une Classe d'Interaction (issue de son SOM) définissant la relation de couplage de sortie (IC et/ou EOC) par *publishInteractionClass*(). D'un autre coté, un modèle G-DEVS possédant un port d'entrée influencé « souscrit » à la Classe d'Interaction (IC et/ou EIC) publiée par le port qui l'influence, en utilisant *subscribeInteractionClass*(). Le FOM contiendra en conséquence toutes les relations de couplages entre les modèles DEVS/G-DEVS du modèle couplé distribué, ces informations seront partagées sous forme de classes d'interaction. Nous pouvons aussi produire, dans le FOM, une classe d'objet qui définit les éléments partagés des simulations locales. Ces objets permettent, par exemple, de suivre l'évolution des valeurs de variables d'état d'un modèle, nous reviendrons dans le chapitre 4 sur l'utilité de ces objets en particulier pour une fédération ne contenant pas uniquement des fédérés modèles G-DEVS. Dans ce cas, les fédérés souhaitant publier un suivi de ces variables publient avec *publishObjectClassAttributes*() et les fédérés souhaitant souscrire au suivi de ces valeurs utilisent *subscribeObjectClassAttributes*().

Interaction Class Structure Table		
HLAinteractionRoot (N)	CouplingRelation (PS)	ExternalInputCoupling (S)
		ExternalOutputCoupling (P)
		InternalCoupling (PS)

Figure 40 - Table de structure des interactions

Les paramètres des interactions entre fédérés sont définis « TimeStamped Order » (cf. Figure 41) pour respecter le principe de causalité. Ces interactions sont donc émises avec une date associée au temps logique local du fédéré émetteur et sont stockées dans un échéancier du LRC avant d'être délivrées au souscripteur lorsque ce dernier sera temporellement en mesure de traiter ce message.

Parameter Table					
Interaction	**Parameter**	**Datatype**	**Available Dimensions**	**Transportation**	**Order**
CouplingRelation.ExternalInputCoupling	Message Type	HLAASCIIchar	TypeMessage	HLAreliable	TimeStamp
	Transmitter	HLAASCIIstring	NA		
	Event time stamp	HLATimeType	NA		
	Concerned Port	HLAASCIIstring	NA		
	Event dimension	HLAboolean			
	Event Value	HLAopaqueData	NA		
CouplingRelation.ExternalOutputCoupling	Message Type	HLAASCIIchar	TypeMessage	HLAreliable	TimeStamp
	Addressee	HLAASCIIstring	NA		
	Event time stamp	HLATimeType	NA		
	Concerned Port	HLAASCIIstring	NA		
	Event dimension	HLAboolean			
	Event Value	HLAopaqueData	NA		
CouplingRelation.InternalCoupling	Message Type	HLAASCIIchar	TypeMessage	HLAreliable	TimeStamp
	Addressee or Transmitter	HLAASCIIstring	NA		
	Event time stamp	HLATimeType	NA		
	Concerned Port	HLAASCIIstring	NA		
	Event dimension	HLAboolean			
	Event Value	HLAopaqueData	NA		

Figure 41 - Table de paramètres

Lors de la phase d'implémentation des concepts HLA dans chaque fédéré, le programme du fédéré reçoit en héritage la classe *FederateAmbassador*. Cet ajout permettra au fédéré d'utiliser les services du RTI, d'implémenter les services de callback, et de pouvoir notamment faire appel au service de demande d'autorisation de traitement du prochain événement. Cet appel *NextEventRequest*() auprès du RTI permet d'obtenir l'autorisation de traiter le prochain événement local planifié dans le cas d'une simulation dirigée par les événements. Nous avons spécifié précédemment, dans le § 3.2.3 la réaction du DRC à ce type d'appel. Le RTI réagit de façon similaire à une telle demande, à savoir qu'il détermine en fonction du LBTS et des messages à destination de ce fédéré, s'il peut lui accorder le droit de traiter son prochain événement. S'il autorise le fédéré à traiter le message demandé, il envoie le service de rappel *TimeAdvanceGrant()†*, mais si le fédéré possède des messages en attente de réception avant la date de son prochain événement local, le RTI les lui délivre auparavant avec les services de call back *ReceiveInteraction()†* ou *ReflectAttributesValues()†*.

Lorsque le fédéré reçoit une autorisation ou des messages du RTI, il les fait parvenir à ses successeurs. En retour, il reçoit la nouvelle date de ses modèles successeurs et éventuellement un message de sortie qu'il envoie au RTI avec le service *sendInteraction()*. Si les attributs de l'objet partagé « coordinateur local » ont changé, il en informe la fédération avec le service *sendAttributesUpdates()*. Nous développerons les algorithmes à mettre en œuvre dans chaque fédérés pour assurer la communication avec le RTI dans le § suivant de ce chapitre.

La phase cinq consiste en une série d'intégration et de tests. Les fédérés sont exécutés individuellement pour un jeu de données test, à partir d'un ensemble d'événements d'entrées et un ensemble d'événements de sorties représentatifs.

La phase six consiste en une série de tests. La fédération est exécutée pour un jeu de données test, à partir d'un ensemble d'événements d'entrées et un ensemble d'événements de sorties représentatifs.

Enfin, la septième et dernière étape consiste à fournir une évaluation des résultats fournis par l'exécution de la fédération. Les résultats produits en sortie de la fédération dans son utilisation actuelle sont analysés, discutés et validés quant à leur consistance et leur crédibilité. Si l'on considère que les résultats faillissent à l'un de ces critères, un mécanisme de retour dans le cycle de développement de la fédération est effectué.

Pour résumer, l'intérêt des fédérés « G-DEVS couplés » réside, en particulier, dans leurs capacités de communication distribuée et leurs possibilités d'interfaçage d'une part avec des programmes hétérogènes générateurs d'événements compatibles HLA en amont et d'autre part avec des programmes exploitant les sorties du modèle G-DEVS global en aval.

3.3.3 Algorithme de communication avec le RTI

HLA propose un cadre pour la mise en oeuvre des mécanismes d'échange de messages de façon synchronisée. Ces mécanismes sont en partie implémentés dans le RTI, mais la partie de communication inclue dans les fédérés reste à la charge du concepteur des fédérés et des fédérations. Le concepteur doit ainsi compléter le code de ses simulations avec un algorithme de communication dont la forme doit respecter le cadre défini par la norme HLA.

Considérons un modèle couplé G-DEVS$_{PF}$ que nous souhaitons distribuer. Les modèles composant le modèle couplé conservent localement la structure LCS définie précédemment, qui est intégrée dans un fédéré HLA. Ces fédérés communiquent entre eux au sein d'une fédération au travers du RTI comme décrit dans l'exemple de la Figure 39. Un algorithme de communication, évoqué dans le paragraphe précédent, est implémenté dans les Coordinateurs Locaux qui participent à la Fédération HLA (cf. LCs dans les Ordinateurs 2 et 3 Figure 27); il est responsable de la Gestion du Temps entre le Composant Local et le RTI [FUJIMOTO 98].

Nous présentons donc ici cet algorithme en nous concentrant sur un modèle G-DEVS$_{PF}$ local intégré dans un fédéré HLA. Nous avons proposé dans [ZACHAREWICZ 05a] cet algorithme de communication du fédéré LCS avec ses successeurs hiérarchiques locaux (des Simulateurs) et avec le reste de la fédération au travers du Composant RTI Local comme décrit l'exemple présenté dans la Figure 39. Cet algorithme est illustré Figure 42 en pseudo-code.
Dans ce pseudo-code, nous supposons que le temps logique du fédéré considéré est « Tact » et qu'il possède un événement local suivant planifié dans sa liste d'événements locale à la date « TnextLocal » avec la relation suivante : Tact < TNextLocal. La valeur du Lookahead pour ce fédéré est fixée par la relation suivante (3).

$Lookahead_{Federé} = Min\ D\ (s_{pf}) / s_{pf} \in S_{PF}$ où S_{PF} est l'ensemble des états modèles (3)

Notons que nous mettons en œuvre, similairement aux solutions précédentes, des modèles G-DEVS$_{PF}$ dont la fonction durée de vie des états est définie par la relation (4).

$$D(s_{pf}) = D(Ph_i) = \text{constante} > 0 \quad (4)$$

$$/\ s_{pf} \in S_{PF} \text{ où } S_{PF} \text{ est l'ensemble des états du modèle}$$

$$\& \ Ph_i \in \text{PHASE où PHASE est l'ensemble des phases du modèle}$$

L'aspect novateur de cet algorithme est apporté par l'appel au service du RTI *queryLITS*() qui fournit au fédéré son LITS[44]. La relation ci dessous (5) rappelle la fonction définie par [FUJIMOTO 98]. Le fédéré est ainsi sûr d'avancer dans le respect de la causalité jusqu'à la valeur de LITS.

$$LITS_{Fédéré} = \text{Min}(Lookahead_i, \text{Message dans la queue du RTI non encore livré}) / i \in I \quad (5)$$

Où I est l'ensemble des Fédérés influenceurs

L'utilisation du service *NMRA*() (*NextMessageRequest*() HLA version 1516, équivalent à *NERA*() HLA version 1.3) du RTI, implique la définition d'une durée ε négligeable comparée aux autres durées utilisées dans la simulation. En effet, le rappel de service *TimeAdvance-Grant*(T)†, en réponse à l'appel *NMRA*(T), ne suppose pas qu'un fédéré ait reçu tous les messages associés à une date donnée T. Pour s'assurer que le fédéré ait reçu le jeu complet d'événements à T, suite à un appel avec *NMRA*(T), il doit obtenir un octroi d'avancement supplémentaire à la date T', conformément à la relation temporelle (6).

$$T' = T + \varepsilon > T. \quad (6)$$

L'avance de temps à T' garantit que tous les événements associés à la date T ont été reçus.
L'utilisation du service *queryLITS*() du RTI permet de préserver une valeur de lookahead des fédérés DEVS non nulle dans le traitement de nombreux messages locaux, en fonction du comportement de livraison des messages et de la rapidité de simulation des influenceurs. Ce résultat est significatif, en accord avec les postulats des travaux de [FUJIMOTO 88], qui stipulent que l'utilisation d'un Lookahead positif augmente la performance d'une simulation distribuée. Cette solution libère ainsi les fédérés influencés de la contrainte de leurs influenceurs pour une période égale au Lookahead. Elle augmente alors le parallélisme de la simulation globale. Nous reviendrons dans le chapitre suivant sur les gains de performance mesurés dans le cadre de la mise en œuvre de cet algorithme.

[44] Least Incoming Time Stamp

```
Ask for Join-Federation-Request (Coordinator, EIC, EOC, IC)
Do queryLITS()
       If (TNextLocal ≤ LITS)
            ComputeOutput() // associated to next local internal event timestamped TNextLocal
            SendInteraction(TnextLocal) // send output without reducing federate Lookahead
            NMRA(TNextLocal) // RTI 1516 NextMessageRequestAvailable(TNextLocal) and then wait for RTI
                             answer
       Else // if (TNextLocal > LITS)
            NMRA(TnextLocal - Lookahead)
            WaitUntil(RTI responds callback)
            If (TimeAdvanceGrant (TNextLocal - Lookahead))
                queryLITS()
                If ((TNextLocal) > LITS)
                    ModifyLookahead(zero)
                Else // If ((TNextLocal) ≤ LITS)
                    ComputeOutput() // associated to next local internal event timestamped TNextLocal
                    SendInteraction(TnextLocal) // send output without reducing federate Lookahead
                NMRA(TNextLocal) // then wait for RTI answer
            Else If (ReceiveInteraction(T'≤ (TnextLocal - Lookahead)) & TimeAdvanceGrant(T'))
                      Do
                             NMRA(T'+ε) // guaranty to have received simultaneous event timestamped T'.
                             WaitUntil(RTI responds callback)
                             If (TimeAdvanceGrant(T'+ε))
                                  ComputeExternalTransition() //        associated to external event(s)
                                                                       timestamped T'
                                  Break to beginning // with new TnextLocal.
                             Else If ((ReceiveInteraction(T') & TimeAdvanceGrant(T'))
                                       AddtoSimultaneousMessageList()
                      While (TimeAdvanceGrant(T') < T'+ε)
       WaitUntil(RTI responds callback)
       If (TimeAdvanceGrant(TNextLocal))
             If (output not already sends with positive Lookahead)
                  ComputeOutput() // associated to next local internal event timestamped TNextLocal
                  SendInteraction(TNextLocal) // send output with zero federate Lookahead
             ComputeInternalTransition() // associated to internal event timestamped TNextLocal
             Do
                  NMRA(TNextLocal+ε) // guaranty to have received simultaneous event timestamped
            TNextLocal.
                  WaitUntil(RTI responds callback)
                  If (TimeAdvanceGrant(TNextLocal+ε))
                       ComputeExternalTransition() //  associated to eventual external event(s) time-
                                                      stamped T
                       ModifyLookahead(min of D(S))
                       Break to beginning
                  Else if ((ReceiveInteraction(T') & TimeAdvanceGrant(T'))
                            AddtoSimultaneousMessageList()
             While (TimeAdvanceGrant(TNextLocal) < TNextLocal +ε)
       Else If ReceiveInteraction(T<TNextLocal) & TimeAdvanceGrant(T)
                  Do
                       NMRA(T+ε) // guaranty to have received simultaneous event timestamped T.
                       WaitUntil(RTI responds callback)
                       If (TimeAdvanceGrant(T+ε))
                              ComputeExternalTransition() //    associated to external event(s) time-
                                                               stamped T
                              Break to beginning // with new TnextLocal.
                       Else if ((ReceiveInteraction(T') & TimeAdvanceGrant(T'))
                                 AddtoSimultaneousMessageList()
                  While (TimeAdvanceGrant(T) < T +ε)
While (Simulation not end)
```

Figure 42 - Algorithme de communication avec le RTI

Pour simplifier le pseudo-code de la Figure 42 et distinguer explicitement l'utilisation des fonctions *ComputeExternalTransition*(), *ComputeInternalTransition*() et *ComputeOutput*(), nous les présentons dans cette figure au niveau du LC. Dans la pratique, nous utiliserons une fonction nommée *SendMessageToSimulator*() au niveau du LC et traiterons ensuite ces messages au niveau du Simulateur.

Pour conclure, cet algorithme présente la partie de communication à mettre en œuvre dans chaque LC de chaque fédéré. Cet algorithme repose sur l'utilisation d'un Lookahead positif, dans le cas où les modèles G-DEVS locaux sous-jacents possèdent au moins un état avec une durée de vie nulle ; son efficacité est alors réduite à celle de l'algorithme proposé par [ZEIGLER 99].

3.4 Amélioration du calcul du Lookahead dans G-DEVS/HLA

En simulation distribuée, le Lookahead est le délai minimum d'un modèle pour émettre un événement de sortie à partir d'un événement reçu. Ce retard correspond pour les modèles G-DEVS à la prochaine exécution de $\lambda(s)$, associée à la plus courte **D**(s) qui peut s'écouler.

Dans les deux derniers algorithmes d'intégration proposés par [LAKE 00] et [ZACHAREWICZ 05a], présentés dans § 3.3.3, le Lookahead est fixé égal à la durée de vie minimale **D**(s) parmi tous les états du modèle présenté dans les relations (3) et (4). En effet, dans les modèles G-DEVS, les événements de sortie sont associés aux transitions internes qui se produisent à l'instant où la durée de vie de l'état actuel est écoulée. Donc, dans le pire des cas, le premier événement pouvant être émis, est borné inférieurement par la durée de vie minimale parmi tous les états. Dans le cas d'un modèle G-DEVS possédant au moins un état avec une durée de vie nulle, la valeur minimale du Lookahead HLA de ce modèle est, par conséquence, également nulle, ce qui restreint l'efficacité de l'algorithme. Pour améliorer le calcul du Lookahead, nous pouvons considérer comme pire des cas, la fonction de sortie $\lambda(s)$ de la phase, accessible par une séquence de δ_{ext}, qui contient la durée de vie minimale.

3.4.1 Application du calcul du Lookahead à la classe G-DEVS$_{PF1}$

Dans le § 3.3.3, nous avons calculé des valeurs de Lookahead minimales pour les modèles G-DEVS$_{PF}$. Ce calcul peut être affiné en considérant l'état actuel et les fonctions du modèle. Pour répondre à cette attente d'amélioration, nous avons proposé dans [ZACHAREWICZ 05b] un raffinement du calcul du Lookahead pour la sous-classe de modèles G-DEVS$_{PF}$ que nous choisissons de nommer G-DEVS$_{PF1}$ qui respecte la relation d'inclusion (7).

$$\text{G-DEVS}_{PF1} \subset \text{G-DEVS}_{PF} \quad (7)$$

Cette solution permet de fournir une valeur relative du Lookahead en fonction de l'évolution de l'état du modèle. Nous présentons les restrictions de la classe G-DEVS$_{PF}$ dans (8).

$$\text{G-DEVS}_{PF1} = < X, S_{PF1}, Y, \delta_{int}, \delta_{ext}, \lambda, D> \quad (8)$$

avec :

X : ensemble des types des événements externes,

Y : ensemble des types d'événements de sortie

$S_{PF1} = Q_{PF1} \times (A^{n+1})$

$Q_{PF1} = PHASE = \{Ph_1, \ldots, Ph_i, \ldots, Ph_n\} \neq \emptyset / i \in \{\mathbb{N}\}$, ensemble fini.

$A^{n+1} = (a_0 \times ... \times a_{n+1})$ (avec $a_i \in \mathbb{Z} / i \in \{0,\ldots, n+1\}$), représentent le (n+1)-uplet des coefficients du polynôme du dernier événement externe arrivé.

$\delta_{ext}((Ph_i, A^{n+1}), A^{n+1}') = (Ph_i', A^{n+1}')$, Nous ne considérons pas le temps écoulé dans un état pour définir la relation suivante

$\delta_{int}(Ph_i, A^{n+1}) = (Ph_i', A^{n+1}')$,

$\lambda(Ph_i, A^{n+1}) = (A^{n+1}')$ fonction de sortie,

D(S_{PF1}) = D (Ph_i, A^{n+1}) = f_{PF1} (Ph_i), Nous considérons, dans cette première amélioration, des modèles dont chaque valeur de phase est associée à une durée de vie constante comme défini par une fonction f_{PF1}.

La solution présentée ici propose de définir un Lookahead relatif à la phase actuelle du modèle. Pour cela, le Lookahead relatif à un état considéré est calculé en utilisant la liste de ses états successeurs accessibles par des transitions externes. L'étude est restreinte aux états successeurs par transitions externes car ces transitions ne génèrent pas directement d'événement de sortie mais permettent d'atteindre des états qui peuvent en émettre, après écoulement de leur durée de vie qui peut être inférieure à celle de l'état considéré. A partir de l'ensemble des phases atteignables par transition(s) externe(s) depuis Ph_i noté *Succeding_States_List* nous définissons la relation suivante (9).

$$\text{Lookahead } (Ph_i) = \min_{k \in \text{Succeding\_States\_List}} f_{PF1} (Ph_k) \quad (9)$$

Plus en détails, nous pouvons affirmer que le prochain événement de sortie à émettre pour un modèle G-DEVS sera associé à une prochaine transition interne. En conséquence, nous devons trouver la plus rapide transition interne exécutable à partir de l'état actuel du modèle. Pour chaque état, le Lookahead est donc fixé à la durée de vie minimale de la liste de ses états accessibles. Nous proposons, pour trouver cet état à durée de vie minimale atteignable, d'utiliser une recherche sur le graphe des phases pour explorer et déterminer, pour chaque état, tous les états accessibles par une séquence de transitions externes.

3.4.1.1 Recherche de chemin dans le graphe associé à un modèle G-DEVS$_{PF1}$

Il est possible de représenter par un graphe les modèles atomiques G-DEVS$_{PF1}$ (à phase explicites). En effet, soit un graphe G=[X, U] avec X ensemble des sommets et U ensemble des arcs. Les phases sont les sommets (nœuds) du graphe et $\in$ X et les transitions sont les arcs du graphe et $\in$ U. Nous pouvons, en conséquence, appliquer aux graphes associés aux modèles G-DEVS$_{PF1}$ les algorithmes de recherche issus de la théorie des graphes.

L'objectif, décrit dans le § 3.4.1, consiste à déterminer pour un modèle G-DEVS$_{PF1}$l'ensemble des phases atteignables par transitions externes à partir d'une phase désignée, ce problème est une question de connexité classique. [TARJAN 72] a proposé un algorithme de parcours en profondeur (DFS[45]) qui progresse récursivement, à partir d'un sommet S, pour chaque sommet voisin de S. Cet algorithme, contrairement à l'algorithme de parcours en largeur, explore successivement pour chaque sommet les chemins jusqu'aux feuilles, puis il prend le premier sommet voisin jusqu'à ce qu'un sommet n'ait plus de voisins (ou que tous ses voisins soient marqués), et revient alors au sommet père. Si le graphe n'est pas un arbre, l'algorithme peut boucler indéfiniment. Pour pallier à ce problème, la notion de marquage des sommets déjà parcourus a été introduite, on ne parcourt ainsi que les sommets non encore marqués.

[45] Depth First Search

Nous avons défini un algorithme qui explore, à partir d'une phase considérée, le sous-graphe des phases accessibles par une succession de transitions externes pour calculer un Lookahead relatif à l'état actuel. Dans la Figure 43, nous présentons un algorithme en pseudo-code de cette solution basée sur l'algorithme de parcours en profondeur d'abord classique présenté dans le paragraphe précédent. Cet algorithme peut être implémenté itérativement à l'aide d'une pile LIFO[46] contenant les sommets à explorer : on dépile alors un sommet et on empile ses voisins non encore explorés.

```
depth_First_Search (graph, Initial_State)
// Local Variables
x                       //     considered state
Min_lt                  //     minimum life_time value
Succeding_States_List   //     List of reachable states from a considered
                        state by an External Transition
Start
   x <- Initial_State
   Min_lt <- lt(x) // lifetime function
   Succeding_States_List <- Get_Succeding_States(graph, x)
   Do
          If (Existing_Not_Explored_State(Succeding_States_List))
                 x <- First_State_Not_Explored(Succeding_States_List)
                 Succeding_States_List <- Get_Succeding_States(graph, x)
                 Mark_Explored(x)
                 Min_lt <- min(lt(x), Min_lt)
          Else   // x is a leaf or next states are already explored
                     // Go up to 1st preceding state that has not-explored son
                 While (x =! Initial_State &
                                !(Existing_Not_Explored_State(Succeding_States_List)))
                 Do
                        x <- Preceding_State(x)
                        Succeding_States_List <- Get_Succeding_States(graph, x)
                 EndWhile
                 If (Existing_Not_Explored_State(Succeding_States_List))
                        x <- First_State_Not_Explored(Succeding_States_List)
                        Succeding_States_List <- Get_Succeding_States(graph, x)
                        Mark_Explored(x)
                        Min_lt <- min(lt(x), Min_lt)
                 EndIf
          EndIf
   While (x =! Initial_State && !( Existing_Not_Explored_State(Succeding_States_List)))
End
```

Figure 43 - Algorithme de calcul du Lookahead relatif à l'état actuel du modèle DEVS.

Nous utilisons dans cet algorithme la liste d'adjacence d'un noeud considéré du graphe pour obtenir la *Succeding_States_List*. Cet algorithme calcule le Lookahead relatif pour un *Initial_State,* qui est égal à *Min_lt* à la fin de l'exploration de graphe (c'est-à-dire. *Min_lt* est le minimum des durées de vie (lifetime) des états accessibles).

3.4.1.2 Exemple de calcul du Lookahead pour un modèle G-DEVS$_{PF1}$

La Figure 44 représente le graphe associé à un modèle atomique G-DEVS$_{PF1}$ « Commande » dans la représentation graphique de [SONG 95]. Cet exemple nous permet d'illustrer l'utilisation de cet algorithme sur un modèle G-DEVS$_{PF1}$.

[46] Last In First Out

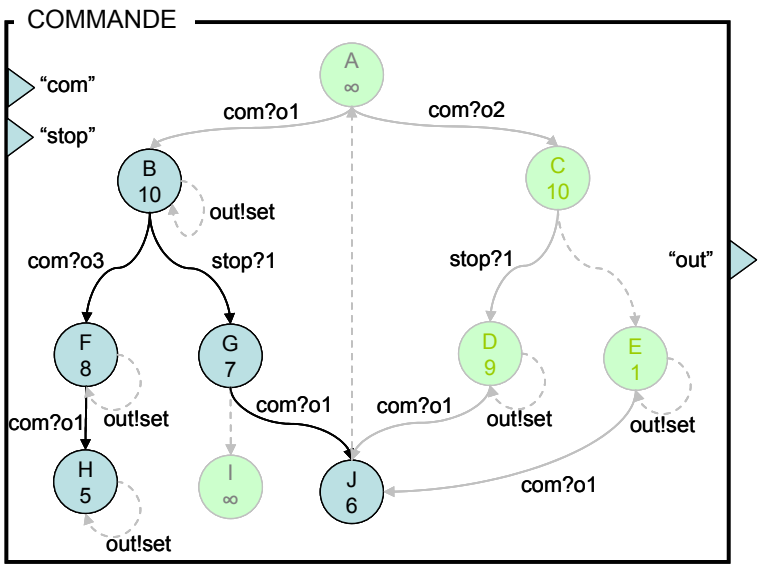


Figure 44 - Exemple de Lookahead relatif à la phase actuelle du modèle G-DEVS$_{PF1}$

L'exemple choisi permet de mettre en évidence le gain, en termes de valeur de Lookahead, lors du passage à un calcul de cette donnée dépendant de l'état actuel du modèle. Le Lookahead absolu de ce modèle est égal à la durée de vie minimale entre tous les états. La phase E comporte la durée de vie minimale du modèle, soit une unité de temps. Le Lookahead absolu de ce modèle, indépendamment de son état actuel lors de la simulation, est donc égal à une unité de temps.

Nous nous focalisons à présent sur chaque phase de ce modèle. Par exemple, dans le cas où la phase actuelle en cours d'exécution est B, la valeur de Lookahead du modèle peut être augmentée. En effet, la phase E ne fait pas parti de l'ensemble des phases atteignables depuis B par une séquence de transitions externes, seules les phases sombres encerclées en noir sur la Figure 44, font partie de cet ensemble. En l'occurrence, la phase de cet ensemble comportant la durée de vie minimale est ici H. L'algorithme proposé nous indique donc que le Lookahead de l'état actuel B peut être augmenté à 5 unités de temps en considérant le calcul de Lookahead relatif à cet état actuel.

Le calcul de toutes les valeurs de Lookahead est réalisé avant d'exécuter la simulation, il n'affecte donc pas la performance de la simulation. Nous devons tout de même garder à l'esprit les restrictions associées à ce calcul. En effet, les classes de modèles G-DEVS$_{PF1}$ autorisées pour ce calcul sont restreintes aux modèles à « phases explicites » et dont la durée de vie de l'état est calculée uniquement à partir d'une seule variable d'état : la phase. Il serait souhaitable d'élargir ces classes de modèles.

3.4.2 Application du calcul du Lookahead à la classe G-DEVS$_{PF2}$

Nous avons proposé, dans le § 3.4.1, une première amélioration du calcul du Lookahead dans le cas de modèles G-DEVS$_{PF1}$ possédant un ensemble fini de valeurs de phases et des durées de vie constantes [ZACHAREWICZ 05b]. Toutes les valeurs de Lookahead relatives à l'état actuel issues de ce calcul sont supérieures (ou dans le pire des cas égales) à la valeur du Lookahead unique.

Nous proposons dans cette section d'étendre la portée de cette amélioration aux classes de modèles (10) dont la durée de vie est définie par plusieurs variables d'état [ZACHAREWICZ 05d].

$$\text{G-DEVS}_{PF1} \subset \text{G-DEVS}_{PF2} \subset \text{G-DEVS}_{PF} \quad (10)$$

Dans la suite, nous considérons les modèles G-DEVS_{PF2} définis par (11), tels que :

$$\text{G-DEVS}_{PF2} = < X, S_{PF2}, Y, \delta_{int}, \delta_{ext}, \lambda, D> \quad (11)$$

avec :

X : ensemble des types des événements externes,

Y : ensemble des types d'événements de sortie

$S_{PF2} = Q_{PF2} \times (A^{n+1})$ avec :

$Q_{PF2} = (PHASE \times B^n)$

Où : $PHASE = \{Ph_1, \ldots, Ph_i, \ldots, Ph_n\} \neq \emptyset / i \in \{\mathbb{N}\}$, ensemble fini.

et $B^n = (b_0 \times ... \times b_n) / b_i \in \mathbb{R}+$ ou $\mathbb{N}$ $/ i \in \{0,\ldots, n\}$, est un ensemble fini (n)-uplet de variables d'état.

$A^{n+1} = (a_0 \times ... \times a_{n+1})$ (avec $a_i \in \mathbb{Z}$ $/ i \in \{0,\ldots, n+1\}$), représentent le (n+1)-uplet des coefficients du polynôme du dernier événement externe arrivé.

$\delta_{ext}((Ph_i, B^n, A^{n+1}), A^{n+1}') = (Ph_i', B^n', A^{n+1}')$, Nous ne considérons pas le temps écoulé dans un état pour définir la relation suivante, Les actions effectuées sur les variables d'état B^n sont des opérations arithmétiques de sommation, soustraction ou multiplication par un scalaire.

$\delta_{int}(Ph_i, B^n, A^{n+1}) = (Ph_i', B^n', A^{n+1}')$,

$\lambda(Ph_i, B^n, A^{n+1}) = (A^{n+1}')$ fonction de sortie,

$D_{PF2}(S_{PF2}) = D(Ph_i, B^n, A^{n+1}) = f_{PF2}(Ph_i, B^n)$, pour chaque phase Ph_i la durée de vie est une fonction mathématique des variables d'états B^n. Cette fonction ne dépend pas de A^{n+1}, qui est uniquement utilisé pour mémoriser les données liées à la dernière transition externe afin de pouvoir les prendre en compte dans la prochaine transition interne.

La Figure 46 a) présente le graphe associé à un modèle appartenant à la classe G-DEVS_{PF2}.

3.4.2.1 Analyse de la fonction durée de vie d'état $D_{PF2}(s)$ d'un modèle G-DEVS_{PF2}

$D_{PF2}(s)$ est une fonction de plusieurs variables à valeurs réelles (14).

$$f(b_1, b_2, ..., b_n) = \alpha_0 + \alpha_1 b_1 + ... + \alpha_n b_n \quad (14)$$

Où α_k = constante $\in \mathbb{R}$, $\forall$ k $\in \{0,\ldots, n\}$

La fonction f doit être continue, définie et dérivable en tous points de la plage de valeurs des variables d'état $(b_1, b_2, ..., b_n)$ qui la définissent pour en déterminer une valeur minimale quant aux valeurs de minimum et maximum de $(b_1, b_2, ..., b_n)$. De plus, nous avons restreint l'étude aux fonctions durées de vies linéaires définies par des variables d'état indépendantes afin de pouvoir décomposer l'étude en une somme d'études de maxima et minima des $(b_1, b_2, ..., b_n)$.

La fonction $D_{PF2}(s)$ est donc une fonction linéaire de plusieurs variables b_1, b_2, ... , b_n dont la forme est exprimée dans l'équation (15). En tenant compte des restrictions sur les paramètres intervenant dans la durée de vie d'état $D_{PF2}(s)$, nous pouvons calculer une valeur minimale pour cette fonction dans chaque phase accessible à partir d'une phase considérée, en prenant pour paramètres une variable d'état phase et les valeurs de minimum et maximum du n-uplet B^n. Plus formellement, pour une phase et des variables d'état b_1,..., b_n, la durée de vie d'état est définie par (15).

$$D_{PF2}((\text{phase}, B^n), A^{n+1}) = \alpha_0 + \alpha_1 b_1 + ... + \alpha_n b_n \qquad (15)$$

Si $\alpha_i < 0$ nous considérons une valeur Maximum de $b_i \ \forall \ i \in \{0,\ldots, n\}$

Si $\alpha_i \geq 0$ nous considérons une valeur Minimum de $b_i \ \forall \ i \in \{0,\ldots, n\}$

En répétant ce calcul pour chaque phase accessible à partir d'une phase considérée, nous calculons le Lookahead (16) de la phase considérée égal au minimum de toutes les durées de vie d'état $D_{PF2}(s)$ des phases accessibles.

$$\text{Lookahead}(Ph_i) = \min_{k \in \text{reachable phase value list of Phi}} \mathbf{D_{PF2}}(Ph_k, \text{min/Max}(B^n)) \quad (16)$$

Si certains modèles G-DEVS$_{PF2}$ fédérés dans une fédération de modèles couplés G-DEVS$_{PF2}$ ne respectent pas la restriction sur la durée de vie d'état $D_{PF2}(s)$ et sur les variations des variables d'état, ou si le calcul de la durée de vie d'état $D_{PF2}(s)$ conclut à un résultat de valeur négative, aucun minimum de durée de vie d'état $D_{PF2}(s)$ ne pourra être calculé. Nous fixons alors le Lookahead du modèle G-DEVS$_{PF2}$ fédéré égal à une valeur minimale ε négligeable devant les autres valeurs prises par la durée de vie d'état $D_{PF2}(s)$. Ainsi, la simulation est contrainte et ralentie par le Lookahead de ce fédéré.

3.4.2.2 Recherche du plus court chemin sur un graphe associé à un modèle G-DEVS$_{PF2}$

Les algorithmes de recherche de chemin sur un graphe issus de la théorie des graphes semblent être les plus adaptés pour analyser les domaines de variation des variables d'état du modèle. En effet les modèles G-DEVS$_{PF2}$ considérés peuvent être représentés par des graphes tels que présentés dans le § 2.4.4 du chapitre 1 (en termes de nœuds/phase et arcs/transitions). De plus, les valeurs des variables d'état sont modifiées par les transitions lors de changements d'états, ces modifications peuvent être assimilées, sur le graphe, au poids de l'arc permettant le passage d'un nœud à un autre.

Cependant, un problème persiste. D'un coté, les algorithmes de recherche de chemin classiques sur un graphe considèrent une seule variable dans le calcul du chemin, cette variable somme le poids (la longueur) du chemin entre deux nœuds. D'un autre coté, les graphes des modèles G-DEVS$_{PF2}$ que nous souhaitons prendre en considération possèdent n+1 variables d'état (la phase et les B^n variables d'état).

Nous proposons une solution à ce problème, qui consiste à décomposer le modèle G-$DEVS_{PF2}$ considéré (par exemple. Figure 46 a) en n sous-modèles, c'est-à-dire pour chaque variable d'état $b_i \in B^n$. Chaque sous-modèle (G-$DEVS_{PF2bi}$) possède donc la définition d'état suivante (12).

$$S_{PF2bi} = ((phase, b_i), A^{n+1}), \qquad (12)$$

A partir de chaque sous-modèle, nous créons un graphe orienté G = [X, U] avec X ensemble des sommets et U ensemble des arcs. Les phases du modèle G-$DEVS_{PF2bi}$ sont représentées comme les nœuds du graphe ($\in$ X) et les transitions δ_{ext} sont les arcs du graphe ($\in$ U). Les Figure 46 b), c) et d) illustrent cette décomposition en sous-modèles.

De plus, nous associons à chaque arc $\delta_{ext} \in U$, un nombre $l(u) \in \mathbb{N}$ appelé « longueur d'arc ». On dit que G est valué par les longueurs l(u) avec u = (i,j) si les nœuds sont i et j. Dans le cas de graphes associés à des modèles G-$DEVS_{PF2}$, nous pouvons considérer que les arcs du graphe associé à un sous-modèle G-$DEVS_{PF2bi}$ sont valués par les actions (voir chapitre 1 § 2.4.4.1) de la fonction δ_{ext} qui manipule la variable d'état $b_i \in B^n$ considérée. La définition de sous-graphes basés sur b_i implique donc que ces variables d'état de B^n soient indépendantes dans l'expression des actions de la transition δ_{ext} du modèle G-$DEVS_{PF2}$. Plus concrètement, les variables b_i doivent être seulement dépendantes de valeurs constantes ou d'elles mêmes dans les fonctions δ_{ext}.

Nous pouvons, à présent, appliquer sur les graphes orientés obtenus, un algorithme de recherche de chemin pour suivre les variations des variables d'état b_i dans l'objectif de trouver le chemin μ (i,j) de la phase i à j dont la longueur totale en terme de b_i est :

$$l(\mu) = \sum_{u \in \mu(i,j)} l(u) \text{ soit minimum (13)}$$

3.4.2.3 Application de l'algorithme de Dijkstra sur le graphe associé à un modèle G-$DEVS_{PF2}$

En considérant un graphe orienté (obtenu d'un sous-modèle G-$DEVS_{PF2bi}$) et une phase, nous déterminons les chemins les plus courts (en terme d'une variable d'état b_j considérée) pour atteindre toutes les autres phases par une séquence de transitions externes δ_{ext}. Cette recherche calcule la valeur minimale de la variable d'état B considérée pour chaque phase accessible, en d'autres termes, le chemin le plus court pour atteindre une $phase_k$ donnée depuis une $phase_i$ en terme d'une variable d'état donné b_j (13, 13b).

$$\min_{(phasei, phasek)} b_j = \underset{\substack{k \in Succeding\_States\_List\ de\ phasei \\ bj = valeur\ initiale\ /\ PHASE = phasei}}{ShortestPath} (phase_i, b_j, phase_k) \text{ (13b)}$$

Pour accéder à cet objectif, nous utilisons l'algorithme de Dijkstra [DIJKSTRA 59]. Cet algorithme classique issu de la théorie des graphes permet de déterminer le plus court chemin entre deux nœuds d'un graphe en termes de poids des arcs. Nous présentons Figure 45 un al-

gorithme de Dijkstra adapté pour le parcours du graphe d'un modèle G-DEVS$_{PF2bi}$ (ne considérant que les transitions externes et une variable d'état b_i). L'algorithme permet de déterminer tous les chemins les plus courts entre les phases reliées par des transitions externes.

```
DijkstraShortestPath (phaseset, nextPhases, distance, phasei, phasek)
// phaseset: Set of model phases reachable by external transition from a given phase
// nextPhases: Set of next Phases from one phase by external transition
// distance: distance between two phases considering the state variable bj
// phasei: source phase
// phasek: target phase

Start
       For (all n in phaseset) // initialization
          n.visited = infinite
          n.precedent = 0
       Endfor
       phasei.visited = 0
       VisitedPhaseList = liste vide
       NotYetVisitedPhaseList = phaseset

       Do (NotYetVisitedPhaseList != liste vide)
          n1 = minimum (NotYetVisitedPhaseList) // phase in NotYetVisitedPhaseList with
                                                   smaller visited value
          VisitedPhaseList.ajouter(n1)
          NotYetVisitedPhaseList.enlever(n1)
          For (all n2 in nextPhases(n1)) // phases linked from n1 by external transition
                 If (n2.visited > n1.visited + distance (n1, n2))
                             // distance is the weight of the transition between n1 et n2
                        in term of bj
                        n2.visited = n1.visited + distance(n1, n2)
                        n2.precedent = n1// to reach n2, need to pass by n1
                 Endif
          Endfor
       While
       path = Empty list
       n = phasek
       While (n != phasei)
          path.addBefore(n)
          n = n.precedent
       EndWhile
       Return path bj
End
```

Figure 45 - Algorithme de Dijkstra pour le parcours du graphe d'un modèle G-DEVS$_{PF2}$.

L'algorithme Dijkstra possède cependant certaines limitations. Notamment, il n'est pas approprié pour les graphes contenant des circuits avec des arcs de poids négatif. En effet, l'itération de tels circuits diminue le poids du chemin $l(\mu)$ jusqu'à donner un longueur négative. Ainsi, les modèles G-DEVS$_{PF2}$ considérés doivent uniquement contenir des variables d'état définies sur $\mathbb{R}^+$ ou $\mathbb{N}$ et des fonctions de transitions externes qui incrémentent les variables d'état b_i. Remarquons que les algorithmes de [WARSHALL 62] et de [FLOYD 62] permettent de rechercher des chemins sur des graphes contenant des arcs de poids négatifs, mais restreignent toujours l'étude à des graphes sans circuit comportant des arcs de poids négatif, ce qui ne résout donc qu'un seul des problèmes.

Nous pouvons également être amené à considérer le plus long chemin d'une phase considérée aux autres (lorsque α_i de b_i est une constante négative), nous utilisons alors l'algorithme Dijkstra modifié (en changeant les valeurs recherchées de minimales à maximales). Cette recherche calcule les valeurs maximales des variables d'état pour chaque phase accessible. Cet algorithme implique également une restriction sur le type de graphe ; il peut être seulement appliqué aux graphes acycliques (c'est-à-dire sans circuits) car il a été démontré que la découverte du chemin le plus long dans un graphe cyclique peut être un problème NP-complexe.

Notons qu'il est possible de considérer des graphes cycliques dans le cas particulier où toutes les durées de vie d'état D(s) ne contiennent aucune sous-fonction ($\alpha_i b_i$) décroissante, puisque dans ce cas, nous ne cherchons pas de valeur maximum des variables d'état.

3.4.2.4 Exemple de calcul du Lookahead pour un modèle G-DEVS$_{PF2}$.

Nous proposons dans la Figure 46 a) un exemple qui illustre l'utilisation de l'algorithme présenté ci-dessus. Le modèle présenté est un G-DEVS$_{PF2}$ d'ordre 1. Cet exemple est ensuite décomposé en sous-modèles G-DEVS$_{PF2bi}$ pour chacune des variables d'état b_1, b_2, b_3 dans les Figure 46 b), c) et d).
Nous considérons un état initial du modèle G-DEVS$_{PF2}$ avec :

Phase = A,
B^2 : $b_1 = 0$, $b_2 = 0$, $b_3 = 0$,
A^1 : $a_0 = 0$ et $a_1 = 0$.

Nous allons calculer, sur l'exemple considéré, le Lookahead de la phase A. En conséquence, nous déterminons les phases accessibles par une séquence de transitions externes, qui sont, pour la phase A de départ, les phases B, C et D. Après avoir isolé les phases atteignables, les valeurs de la durée de vie d'état $\mathbf{D_{PF2}}(s)$ de ces phases doivent être calculées. En effet, les durées de vie ne sont ici plus constantes et associées aux phases comme dans la section précédente, mais elles dépendent, à présent, du « chemin » de transitions externes emprunté pour atteindre la phase considérée.

Pour le calcul du Lookahead de la phase A, nous nous concentrons tout d'abord sur le calcul de la durée de vie d'état $\mathbf{D_{PF2}}(S)$ de la phase D qui est égal à la relation suivante :

$$\mathbf{D_{PF2}}(D, b_1, b_2, b_3) = 3b_1 + b_2 - 2b_3$$

Cela implique donc de considérer la valeur minimale de b_1 et b_2 et la valeur maximale de b_3. En appliquant les algorithmes de Djikstra, on découvre les extremums des variables d'état pour chaque phase.
En considérant uniquement la phase et b_1 dans le graphe du modèle G-DEVS$_{PF2b1}$ de la Figure 46 b), les valeurs minimales de $\mathbf{b_1}$ sont :

10 unités de temps pour la phase B, 2 pour C, 10 pour D et non défini pour E.

Les valeurs minimales de $\mathbf{b_2}$ sur le modèle G-DEVS$_{PF2b2}$, en considérant uniquement la phase et b_2, sont dans le graphe c) de la Figure 46 :

2 unités de temps pour B, 2 pour C, 3 pour D et non défini pour E.

Les valeurs minimales de $\mathbf{b_3}$ sur le modèle G-DEVS$_{PF2b3}$, en considérant uniquement la phase et b_3, sont dans le graphe d) de la Figure 46 :

2 unités de temps pour B, 1 pour C, 2 pour D et non défini pour E.

Les valeurs maximales de $\mathbf{b_3}$ sur le modèle G-DEVS$_{PF2b3}$, en considérant uniquement la phase et b_3, sont dans le graphe d) de la Figure 46 :

2 unités de temps pour B, 1 pour C, 8 pour D et non défini pour E.

En utilisant ces valeurs, nous calculons les valeurs minimales des fonctions **D** (s) pour chaque phase accessible de la phase A :

$$\min \mathbf{D}_{\mathbf{PF2}}(A, \min/\mathrm{Max}\ (B^n)) = b_1 + b_2 + 5 = 0 + 0 + 5 = 5$$
$$\min \mathbf{D}_{\mathbf{PF2}}(B, \min/\mathrm{Max}\ (B^n)) = b_2 + b_3 = 2 + 2 = 4$$
$$\min \mathbf{D}_{\mathbf{PF2}}(C, \min/\mathrm{Max}\ (B^n)) = 2b_1 + b_2 = 4 + 2 = 6$$
$$\min \mathbf{D}_{\mathbf{PF2}}(D, \min/\mathrm{Max}\ (B^n)) = 3b_1 + b_2 - 2b_3 = 30 + 3 - 16 = 17$$

Le Lookahead de la phase A du modèle G-DEVS$_{PF2}$ Figure 46 est donc égal à **D**(s) minimum de toutes les phases accessibles, c'est à dire 4 unités de temps.

En utilisant la même approche, nous pouvons calculer le Lookahead HLA de toutes les phases du modèle. Ces valeurs sont utilisées dans l'algorithme de communication avec le RTI présenté dans [ZACHAREWICZ 05a] et défini dans le § 2.3.

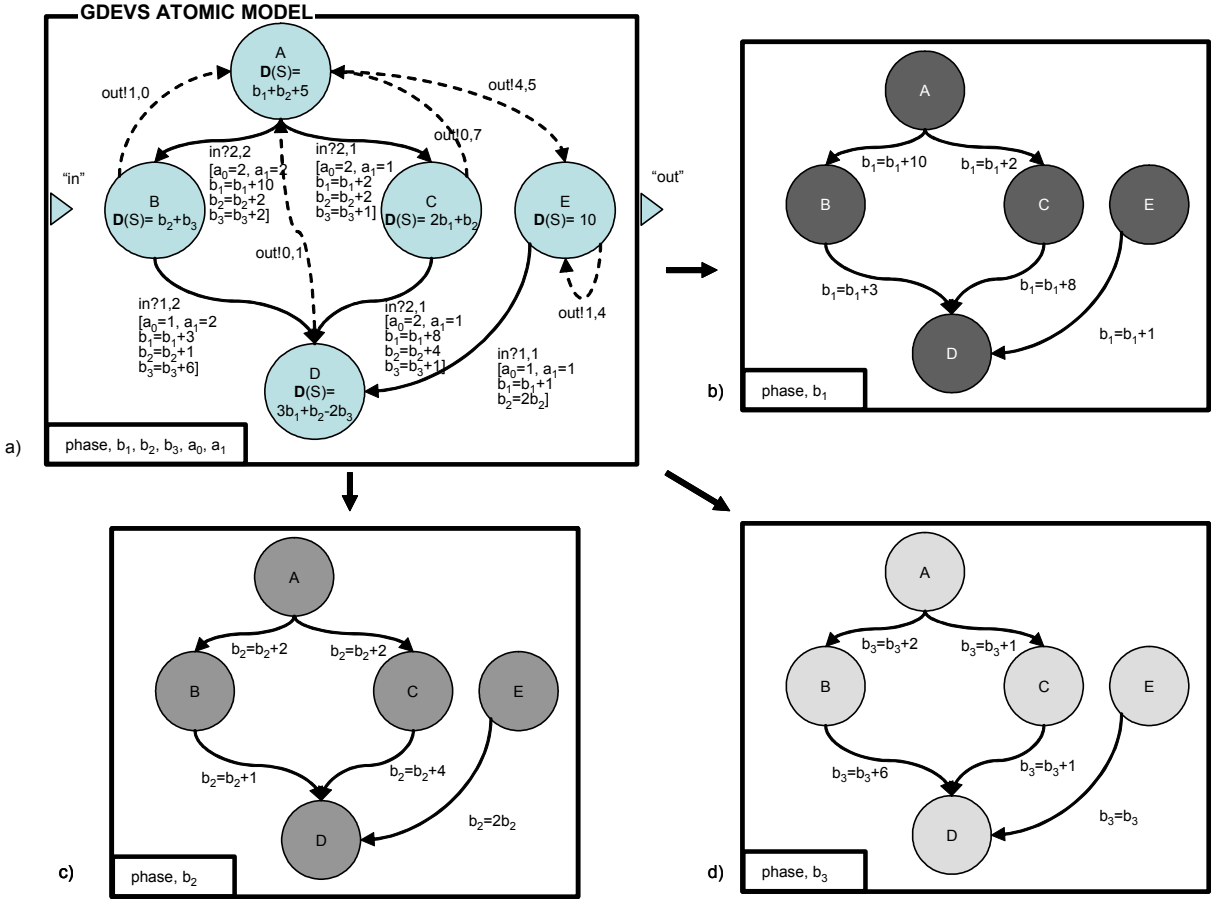


Figure 46 - Exemple de Lookahead relatif à l'état actuel du modèle G-DEVS$_{PF2}$

La limitation de l'application de cette solution dépend notamment du nombre de valeurs de phase dans un modèle ; il ne parait pas judicieux d'essayer de calculer le Lookahead dans un temps raisonnable par rapport à la durée de simulation pour des modèles avec de très nombreuses valeurs de phase et un vecteur B de dimension importante. Il convient, dans ce cas, de comparer la complexité en *O* (m + ln (n)) (avec *m* arcs-transitions et *n* phases-nœuds) de l'algorithme de Dijkstra à celle du simulateur pour déterminer la pertinence du calcul de Loo-

kahead préalable. De plus, cette amélioration ne permet pas d'étendre le calcul du Lookahead à toutes les classes de modèles DEVS/G-DEVS ; par exemple on ne considère pas dans l'étude les modèles avec : des phases non explicites, des variables d'état définies sur $\mathbb{R}$ et des fonctions **D**(s) non linéaires.

4 Perspectives

Après avoir appliqué le calcul de Lookahead aux classes G-DEVS$_{PF1}$ et G-DEVS$_{PF2}$, nous souhaitons continuer à étendre la classe de modèle G-DEVS compatibles HLA sur laquelle peut porter ce calcul de Lookahead.

Pour cela, nous étudions les différentes possibilités pour calculer le plus long chemin sur un graphe valué car ce calcul de chemin est responsable de la restriction sur la classe de modèles G-DEVS considérée dans une simulation distribuée HLA jusqu'à présent. Nous étudions notamment les possibilités offertes par les algorithmes d'approximation du chemin le plus long proposés dans la littérature. Nous envisageons également de considérer d'autres types de fonctions de durée de vie d'état D(s), par exemple les fonctions contraintes dépendantes de plusieurs variables réelles estimées avec l'application du théorème de Lagrange.

5 Conclusion

Dans ce chapitre, nous avons présenté un simulateur conceptuel de modèles G-DEVS qui utilise des modèles mis à plats et distribués.

Nous avons choisi, pour faciliter et formaliser l'intégration des modèles et de leurs simulateurs dans un contexte distribué, de nous conformer à la norme HLA. Nous avons ensuite défini pour chaque fédéré, un nouvel algorithme de communication avec la RTI en effectuant, en particulier, un calcul du Lookahead HLA pour les modèles G-DEVS distribués.

Nous avons également présenté des améliorations du calcul du Lookahead pour des classes particulières de modèles G-DEVS (à phases explicites). Le premier algorithme proposé se base sur la recherche de chemin en profondeur dans un graphe. Il calcule le Lookahead en fonction de la phase actuelle sur le graphe associé aux modèles G-DEVS$_{PF1}$. Le second algorithme se base sur la recherche d'un plus court chemin et s'applique à la classe G-DEVS$_{PF2}$, permettant ainsi de considérer des variables d'état supplémentaires.

Ces algorithmes sont intégrés dans l'environnement de simulation G-DEVS compatible HLA, que nous présentons dans le chapitre suivant.

Chapitre 3. Environnement de simulation G-DEVS / HLA

1 Introduction

Ce chapitre présente un environnement de modélisation et de simulation basé sur le formalisme G-DEVS. Cet environnement repose sur la structure de modèles et sur le simulateur abstrait définis par B.P. Zeigler avec cependant certaines distinctions relatives à la définition de modèles G-DEVS et à l'implémentation du simulateur. De plus, il intègre une interface homme/machine axée vers les utilisateurs souhaitant un haut niveau d'abstraction dans la représentation de leurs modèles. Enfin il comporte une extension permettant de définir des modèles G-DEVS en tant que fédérés HLA.

Le premier apport de cet environnement réside dans la possibilité de décrire des modèles G-DEVS au travers d'une interface graphique. Le deuxième apport se situe dans la définition d'un nouveau moteur de simulation « mis à plat » dont nous montrons que l'algorithme du moteur proposé est moins complexe que celui du moteur hiérarchique. Enfin nous proposons une extension de l'environnement qui permet de simuler des modèles G-DEVS distribués. Dans ce cas, tous les sous-modèles inter-communiquent en échangeant des données au travers d'un réseau local ou par Internet, pour ce faire nous nous conformerons à la norme HLA et utilisons le RTI associé à cette norme pour gérer l'échange de données.

Dans la suite de ce chapitre, nous effectuons une brève présentation des environnements de simulation existants basés sur DEVS. Nous présentons, ensuite, l'environnement de modélisation et simulation nommé LSIS_DME[47] développé au sein du laboratoire LSIS. Pour ce faire, nous introduisons les principales fonctionnalités de cet environnement et leurs utilisations dans le développement de modèles G-DEVS en nous concentrant sur le processus de modélisation. Nous situons ensuite nos travaux au sein de ce projet en présentant le moteur de simulation basé sur les concepts fondamentaux de la théorie DEVS et G-DEVS. Nous développons en particulier la mise à plat des modèles et la mise en œuvre de la compatibilité HLA. Enfin, nous présentons des exemples pour lesquels nous analysons quelques résultats de simu-

[47] LSIS_DEVS Model Editor

lation et conclurons par la présentation de mesures de performances effectuées sur cet environnement.

2 Environnements de modélisation graphique DEVS existants

Gabriel Wainer maintient, sur un site web, une liste actualisée contenant des environnements de modélisation et simulation basés sur le formalisme DEVS [WAINER 04]. Nous présentons ci-dessous deux environnements possédant un outil d'édition graphique de modèles atomiques et couplés DEVS. Ces outils s'adressent à un public de modélisateurs non développeurs permettant ainsi d'utiliser le formalisme DEVS à un haut niveau d'abstraction.

2.1 CD++

CD ++ est un outil développé en C ++, qui permet la définition de modèles DEVS atomiques et Cell-DEVS [WAINER 02]. Cet outil permet également la définition de modèles DEVS et Cell-DEVS couplés en utilisant un langage de spécification de haut niveau. Différentes versions sont proposées incluant des simulateurs à temps réel, parallèles ou centralisés. Cet outil a été développé par Gabriel Wainer et ses étudiants, en partenariat avec les universités de Carleton (Canada), et Buenos Aires (Argentine).

2.2 JDEVS

JDEVS permet d'utiliser des modèles à événements discrets, dans un but général. Il peut être utilisé pour des représentations objet, la modélisation basée sur les composants, le développement de modèles de simulation connectés avec des SIG[48], ou des systèmes collaboratifs. Il intègre un outil de modélisation et d'exécution visuel. Il a été développé par l'équipe de Jean-Baptiste Filippi de l'Université de la Corse à Corte. [FILIPPI 04]

2.3 Analyse

Néanmoins, ces environnements complets ne possèdent pas, pour la majorité d'entre eux, la capacité de définir d'autres variables d'état que la phase. En effet l'amalgame est souvent fait entre phase et état. De plus, ils ne permettent pas de définir des événements contenant une liste de valeurs réelles nécessaires pour la définition d'un modèle G-DEVS. Enfin, il n'existe pas d'outils répondant aux critères précédents et permettant d'effectuer des simulations distribuées de modèles DEVS ou G-DEVS.

3 Spécification de l'environnement LSIS_DME

Nous avons vu précédemment que les environnements de modélisation et de simulation basés sur la définition graphique des modèles atomiques et couplés DEVS ne permettent pas la création de modèles DEVS / G-DEVS à plusieurs variables d'état. De plus, il n'existe pas d'outil permettant la transformation d'un modèle DEVS en fédéré HLA. Pour répondre à ces

[48] Système d'Information Géographique

carences, le LSIS a développé un outil [LSIS 05] pour les modélisateurs non experts du formalisme DEVS / G-DEVS et d'HLA.

3.1 Contributions

Ce projet est le résultat du travail d'un groupe de spécificateurs, concepteurs et de développeurs du LSIS au cours des années 2004-2006. Le logiciel a été développé en Java avec l'outil de développement JBuilder 9 Entreprise [BORLAND 02] puis Eclipse [ECLIPSE 04]. Il peut être dégagé deux axes de travail dans ce projet : la partie modélisation graphique et la partie simulation.

Mes contributions dans la partie modélisation du projet LSIS_DME ont consisté à définir formellement la classe de modèles G-DEVS$_{PF}$ présentée dans le premier chapitre et à spécifier le modèle graphique G-DEVS atomique et couplé qui en découle dans LSIS_DME. La phase de développement de cette partie a été assurée par Frédéric Châne et par Olivier Ricignuolo ingénieurs au LSIS.

Mes contributions dans la partie simulation du projet LSIS_DME, ont consisté principalement à spécifier, concevoir et développer le composant simulateur de modèles G-DEVS$_{PF}$, le composant coordinateur séquentiel (comportant une fonction de mise à plat des modèles) et le composant coordinateur distribué (comportant également une fonction de mise à plat de modèles, des fonctions de communications avec le RTI HLA, et des fonctions permettant de calculer le Lookahead des modèles G-DEVS$_{PF1}$ et G-DEVS$_{PF2}$) (cf. chapitre 2 § 3.4.1 3.4.2).

3.2 Spécification de l'éditeur Graphique LSIS_DME

L'éditeur LSIS_DME intègre un outil de modélisation graphique. Cet outil permet la création, le stockage en bibliothèque, la modification et la composition de modèles. L'éditeur permet de travailler visuellement avec des formes géométriques représentant les différents éléments constituant un modèle DEVS / G-DEVS atomique ou couplé. L'interface graphique obtenue est le résultat de l'utilisation de la bibliothèque graphique Java JGraph [JGRAPH 06] qui propose la réutilisation d'objets graphiques prédéfinis.

3.2.1 Edition de modèles atomiques

L'éditeur de modèles atomiques de LSIS_DME repose sur la notation présentée dans le § 2.4.4.1 du chapitre 1, proposée par [SONG 94].

Fondamentalement, l'utilisateur doit définir l'ensemble des phases de son modèle atomique, auxquelles il associe un nom et une durée de vie. Il peut ensuite interconnecter les phases avec des transitions internes et externes en fonction du comportement souhaité. Enfin, il doit définir les ports d'entrée et de sortie de son modèle, en définissant leurs domaines de valeurs recevables. Les ports comportent un type d'événement recevable mais aussi un degré correspondant au degré du polynôme représenté dans l'événement. A titre d'exemple, ce degré est égal à zéro pour un modèle DEVS et supérieur à zéro pour tout autre modèle G-DEVS.

La Figure 47 présente un exemple de modèle DEVS atomique nommé « Commande », Il possède cinq phases, quatre transitions internes et trois transitions externes. Ce modèle possède également des ports d'entrée et de sortie.

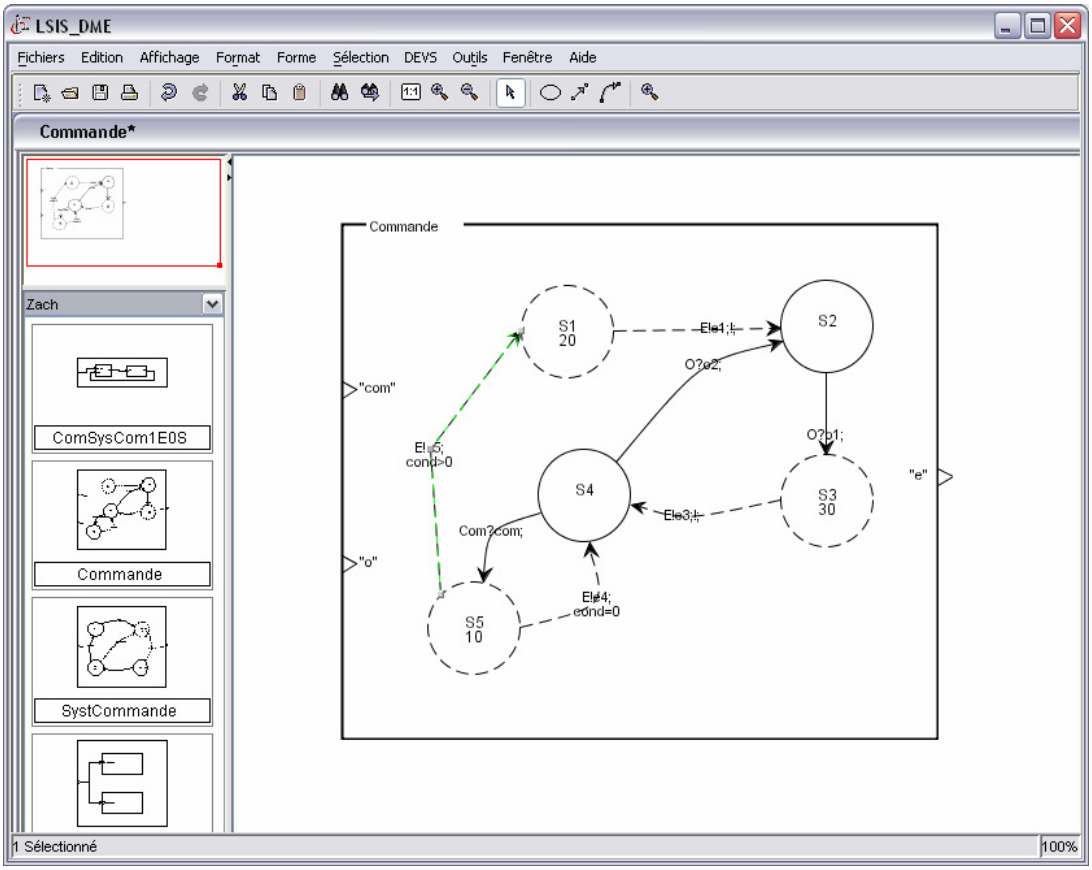


Figure 47 - Edition d'un modèle atomique graphique avec LSIS_DME

Il est important ici de rappeler que les modèles créés graphiquement sont nommés « à phase explicite », dont nous avons présenté la classe G-DEVS$_{PF}$ dans le § 2.4.4 du chapitre 1. Dans l'absolu un état de modèle G-DEVS$_{PF}$ est défini par la phase et par les ensembles de valeurs des variables d'état V et A^{n+1}. Une lacune classique des éditeurs de modèles graphiques DEVS est due à l'impossibilité de définir des variables d'état autres que la phase (ensemble V = Ø dans ces modèles).

En réponse à cette limitation, l'éditeur de modèles de LSIS_DME inclue la possibilité de manipuler d'autres variables d'état que la phase couramment utilisée dans les modèles graphiques. En effet, l'outil prévoit la possibilité de définir, dans les modèles atomiques, des variables d'état (appartenant à l'ensemble V de G-DEVS$_{PF}$) à l'aide d'une fenêtre d'édition associée au modèle. Les valeurs de ces variables d'état peuvent être utilisées comme conditions de transition, et sont modifiées par les actions exécutées lors des transitions d'état (voir @cond et []action chapitre 1 Figure 7). Dans l'exemple Figure 47, deux arcs de transitions internes sont définis en sortie de la phase S5, un premier en direction de S4 et un deuxième en direction de S1. A partir de la phase S5, un choix de transition interne doit donc être effectué, il repose sur une condition portant sur une variable d'état nommée *Cond* ∈ V.

Enfin, ces modèles atomiques sont simulables et/ou stockables en bibliothèque pour une utilisation ultérieure, pour modification ou utilisation dans un modèle couplé.

3.2.2 Edition de modèles couplés

3.2.2.1 Modèles couplés DEVS

Un modèle couplé se construit à partir d'une bibliothèque de modèles précédemment créés. Nous avons retenu la représentation graphique introduite par [CARSTEN 94] rappelées dans le § 2.4.5 du chapitre 1.

L'outil permet, par *Drag and Drop*, de sélectionner des modèles en bibliothèque et de les ajouter en tant que modèles composants du modèle considéré. L'outil permet ensuite de définir les relations de couplages entre les composants du modèle. Les relations de couplages sont des arcs éditables. Il est possible de donner un nom à chaque connexion.

De même, il est possible de créer des ports d'entrée et de sortie sur le modèle. La cohérence du type des événements échangés par couplage entre les ports sera vérifiée, en relation avec le type de ports des sous-modèles connectés, lors de la sauvegarde du modèle ou de l'initialisation de sa simulation.

La Figure 48 présente un modèle couplé nommé « ComsysCom1EOS ». Ce modèle est composé de deux sous-modèles « Commande 0 » et SystCommmande 0 » provenant de la bibliothèque « Zach ». La fenêtre graphique de LSIS_DME donne une représentation des éléments de la bibliothèque active en miniatures dans le panneau vertical gauche sur la Figure 48. Ce modèle comporte deux couplages internes et un couplage d'entrée externe.

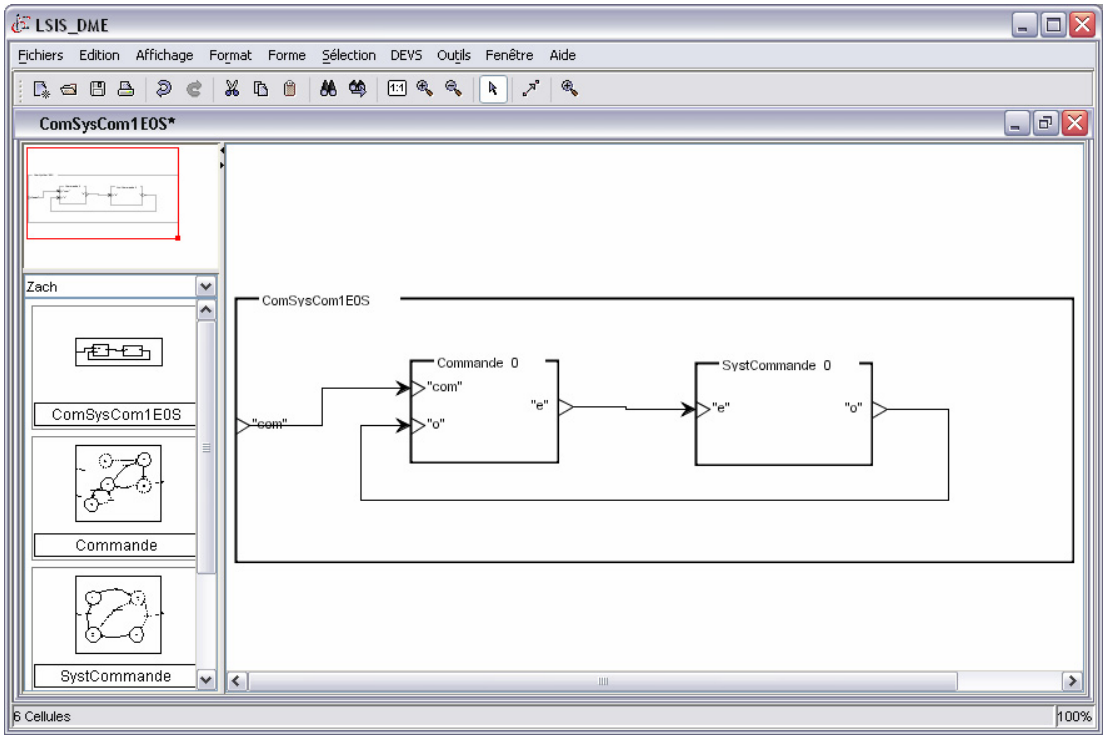


Figure 48 - Edition d'un modèle couplé graphique avec LSIS_DME

3.2.2.2 Spécificités des modèles couplés G-DEVS

L'éditeur permet de modéliser des modèles DEVS ou G-DEVS. Dans le cas d'un modèle G-DEVS, un type d'événement spécifique est prévu (cf. un vecteur de valeurs représentant les coefficients du polynôme approximant le signal original).

Dans le cas de modèles G-DEVS, il est également nécessaire de définir une fonction de couplage des modèles d'ordres différents. En effet, le couplage de modèles G-DEVS utilisant des événements d'ordres différents nécessite une extrapolation ou une réduction des informations contenues dans les messages circulant au sein de ce type de modèles couplés.

[ESCUDE 00] propose une fonction qui définit trois transformations possibles : Expansion, Compression, Direct. Les transformations par expansion et directes sont triviales. La compression est plus complexe, et peut être appréhendée d'un point de vu spatial (une seule approximation) ou temporel (décomposition en plusieurs événements). En pratique, les auteurs ont défini un modèle atomique nommé *Joiner* qui comporte la fonction de transformation afin d'adapter le niveau du message émis à un niveau recevable par le modèle influencé. Ce modèle est issu d'une bibliothèque spécifique contenant les modèles de couplage pour interfacer les modèles couplés G-DEVS manipulant des événements de niveaux différents.

3.3 Spécification de l'environnement de simulation LSIS_DME

Au delà de notre participation à la spécification de l'éditeur graphique, nos travaux, dans le projet LSIS_DME, se sont principalement axés sur la partie simulation des modèles générés par l'éditeur, et plus précisément sur la manipulation des modèles stockés et leur utilisation par le moteur de simulation.

Le moteur de simulation de l'environnement LSIS_DME se base sur l'architecture de simulation proposée par B.P. Zeigler dans [ZEIGLER 00], dont nous proposons une adaptation visant à réduire la structure hiérarchique de simulation, de plus une extension a été définie pour simuler les modèles de façon distribuée, en respectant la norme HLA.

3.3.1 Fonctions de mise à plat de la structure hiérarchique de modèles

Pour effectuer la transformation de mise à plat des modèles en vue de leurs simulations, deux solutions sont envisageables.

La première solution consiste à conserver le format de stockage des modèles couplés avec toute la hiérarchie des modèles. Lors de l'initialisation d'une simulation, l'environnement effectuera un parcours de la structure arborescente du modèle considéré pour récupérer les modèles atomiques qui se trouvent sur les feuilles. Cette solution présente l'avantage de s'appliquer à tout modèle couplé mais elle demande de mettre en oeuvre un algorithme de parcours d'arbre qui peut être lent dans le cas d'un modèle couplé complexe.

La deuxième solution consiste à effectuer une transformation de mise à plat à chaque sauvegarde d'un modèle ou lors du lancement de sa simulation. Dans ce cas, les modèles considérés contiennent au maximum deux niveaux hiérarchiques puisque les modèles inclus proviennent de la librairie et qu'ils ont subit une mise à plat préalable lors de leurs sauvegar-

des. Cette solution permet de mettre en oeuvre des algorithmes de parcours moins complexes ; en contrepartie tous les modèles inclus doivent avoir étés mis à plat précédemment. Nous avons choisi de retenir la deuxième solution pour des raisons de complexité algorithmique moindre, et de rapidité d'exécution dans les algorithmes de parcours des modèles et des couplages.

La solution retenue pour la sauvegarde des modèles consiste donc à archiver d'une part les modèles hiérarchiques (pour l'édition et la composition de modèles) et d'autre part les modèles non hiérarchiques (pour la simulation). Les modèles stockés en bibliothèque possèdent donc tous une structure hiérarchique et une structure équivalente non hiérarchique (cf. Figure 49). Le choix de ce type de stockage permet de toujours manipuler des modèles qui possèdent une version équivalente comportant au plus un niveau hiérarchique. Pour résumer, lors de l'enregistrement d'un modèle ou du lancement d'une simulation, le modèle pris en paramètre sera donc composé de modèles atomiques mais également de modèles couplés issus d'une librairie. La transformation consistera donc en la mise à plat du seul niveau hiérarchique des modèles couplés inclus dans le modèle considéré.

Nous présentons, dans la section suivante, la structure de données des modèles qui est manipulée par cet algorithme. Par la suite, nous introduirons, l'algorithme de mise à plat des modèles. Cet algorithme de mise à plat est composé de deux fonctions : la première met à plat les modèles hiérarchiques et la deuxième permet de définir les relations de couplages entre les modèles plats.

3.3.1.1 Diagramme de classes des modèles de LSIS_DME

Le diagramme de classes du modèle de données spécifié pour LSIS_DME, présenté Figure 49, est basé sur la classe présentée par [ZEIGLER 95] (cf. chapitre 2 Figure 21). Il intègre cependant certaines spécificités propres à l'édition graphique de modèles et à la mise à plat de modèles pour la simulation.

Structure de classes du modèle atomique LSIS_DME

Le modèle de classes relatif au modèle atomique DEVS de LSIS_DME détaille en particulier l'attribut `phases` ainsi que l'attribut `OtherStateVariablesSet` permettant de représenter les autres variables d'état qui définissent l'état du modèle. Il comporte également un attribut `graphicalData` utilisé pour représenter les informations relatives à la représentation graphique du modèle. Cet attribut n'a qu'une utilité dans la partie modélisation et réutilisation des modèles. Il présente également les différentes fonctions définies par le formalisme DEVS. Enfin, il possède un attribut `eventOrder` définissant le degré du G-DEVS modélisé.

Structure de classes du modèle couplé LSIS_DME

La Figure 49 présente, en particulier, la classe du modèle couplé dans LSIS_DME. Cette classe comporte une liste de ports influents `influentPortListWithHierarchy`, définissant les ports influents du modèle, chacun de ces ports faisant, à leurs tours, référence à une liste

de port influencés `InfluencedPortList`. Par rapport à la représentation originale de [ZEIGLER 95] présenté dans le chapitre 2 Figure 21, cette classe possède la particularité, de contenir des attributs `includedModelWithoutHierarchyList` et `nonHierarchicalInfluentPortList` décrivant un modèle couplé non hiérarchique équivalant au modèle créé par l'utilisateur de l'outil. En effet, lors de l'enregistrement du modèle couplé en librairie ou lors du lancement de la simulation, l'environnement effectue une mise à plat du modèle. Ces données sont stockées dans des objets de type List.

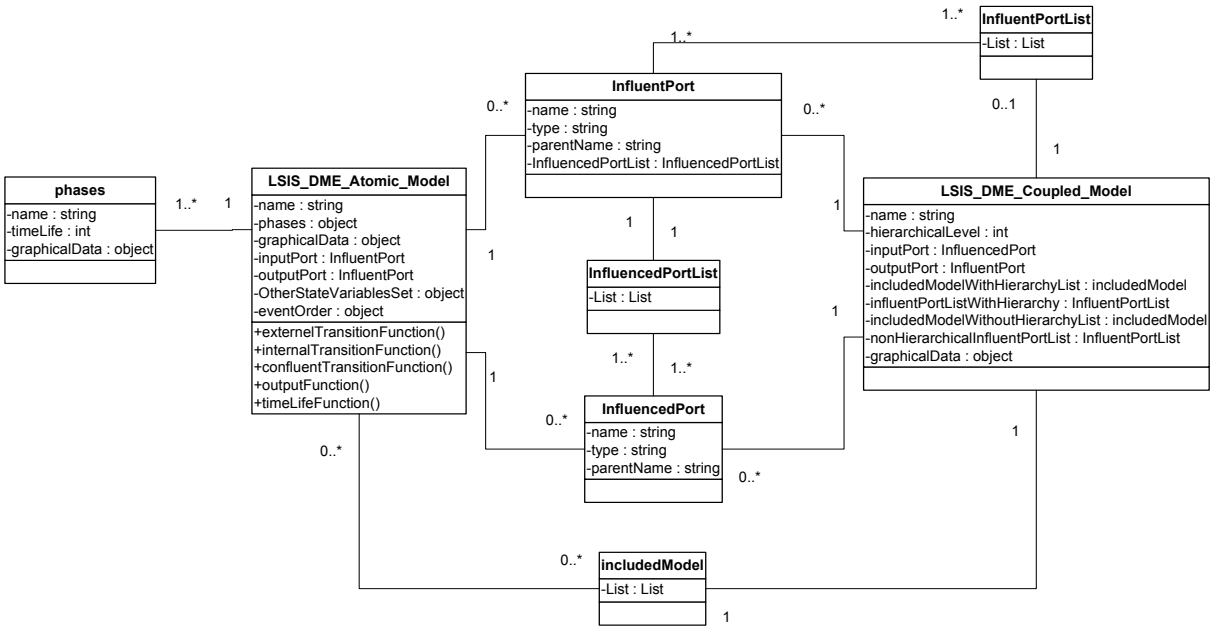


Figure 49 - Classes du modèle couplé DEVS dans LSIS_DME

3.3.1.2 Fonction de transformation des modèles

Pour présenter la fonction de mise à plat, nous nous focalisons sur certains attributs de la structure de données du modèle couplé de LSIS_DME présentée dans la Figure 49. La classe modèle couplé possède notamment un attribut contenant l'ensemble des modèles couplés **hiérarchiques** inclus, nommé `includedModelWithHierarchyList`. Cet attribut contient lui-même un ensemble d'`includedModel` (modèle atomique ou couplé). Cette classe contient, de plus, un attribut nommé `includedModelWithoutHierarchyList`, contenant une liste `includedModel` de modèles **non hiérarchiques** (atomiques). Lors de la création d'un modèle, cette deuxième liste est initialement vide.

Le pseudo code exposé ci-dessous dans la Figure 50 spécifie la fonction `flatteningModels` de LSIS_DME. Cette fonction a pour objectif de générer l'ensemble des modèles atomiques `includedModelWithoutHierarchyList` à partir de l'ensemble des modèles hiérarchiques `includedModelWithHierarchyList` d'un modèle. Cette fonction est donc appelée au moment de l'enregistrement d'un modèle en bibliothèque ou lors d'une initialisation précédant l'exécution d'une simulation. L'objectif de la fonction `flatteningModels` est de parcourir l'ensemble `includedModelWithHierarchyList` ; pour chaque `includedModel`, un test est effectué. Si ce sous-modèle est atomique, il est copié dans `includedModelWithou-`

tHierarchyList. Si ce sous-modèle est couplé, tous les modèles includedModelWithoutHierarchy' contenus dans l'attribut includedModel.includedModelWithoutHierarchyList de ce sous-modèle sont copiés dans l'attribut includedModelWithoutHierarchyList du modèle considéré.

```
includedModelWithoutHierarchy flatteningModels (consideredCoupledModel)
for (all includedModel in consideredCoupledModel.includedModelWithHierarchyList)
   if (includedModel.hierarchicalLevel == 0) // no hierarchy in this model
            includedModelWithoutHierarchyList add (includedModel)
   else // the included model is hierarchical
   for (all includedModelWithoutHierarchy' in includedModel.includedModelWithoutHierarchyList)
            includedModelWithoutHierarchyList add (includedModelWithoutHierarchy')
```

Figure 50 - Fonction de mise à plat des modèles dans LSIS_DME

Pour résumer, tout modèle stocké dans la bibliothèque de modèles est donc décrit de la façon suivante : il comporte un ensemble de modèles hiérarchiques et un ensemble de modèles non hiérarchiques, tous deux complétés. Les modèles contenus dans l'ensemble des modèles non hiérarchiques ne sont pas modifiés, ils sont uniquement copiés (eux ou leurs sous-modèles) dans l'ensemble des modèles non hiérarchiques. En effet l'ensemble includedModelWithHierarchyList continu à être utilisé pour la partie modélisation, qui reste une représentation modulaire et hiérarchique.

La mise à plat d'un modèle nécessite également une transformation du couplage des modèles inclus. En effet, les relations de couplage du modèle considéré doivent faire référence uniquement aux modèles atomiques du modèle non hiérarchique et non plus aux modèles couplés de haut niveau de la forme hiérarchique.

3.3.1.3 Fonction de transformation du couplage de modèles

Le pseudo code exposé Figure 50, considère la fonction CouplingTransformation de l'environnement. Cette fonction a pour objectif de générer un ensemble de relations de couplages entre modèles atomiques, à partir de l'ensemble des relations de couplages de la structure hiérarchique d'un modèle.

La première partie de l'algorithme permet de définir les ports influents du modèle non hiérarchique (à partir de la structure influentPort, cf. Figure 51) qui seront insérés dans la liste nonHierarchicalInfluentPortList du modèle.

Chaque influentPort de la liste InfluentPortList est donc analysé. S'il a pour parent le modèle considéré, cette structure sera copiée directement dans une liste intermédiaire IntermediaryInfluentPortList, qui contient toutes les relations de couplage dont les ports influents ont été modifiés. Si l'influentPort a pour parent un modèle inclus, le contenu de ce modèle inclus doit alors être étudié pour déterminer les sous-modèles influenceurs de l'influentPort considéré et créer un newInfluentPort pour chaque port l'influençant. La partie concernant les ports influencés de l'influentPort est recopiée dans chaque newInfluentPort. Chaque newInfluentPort est ajouté à IntermediaryInfluentPortList.

La deuxième partie de l'algorithme parcourt la liste `IntermediaryInfluentPortList`. Chaque port influencé (`influencedPort'`) issu de la liste `influencedPortList` de chaque port influent `influentPort'` doit être analysé. Si la structure `influencedPort'` a pour parent le modèle considéré ou si elles ont pour parent des modèles atomiques, une simple copie sera effectuée dans la liste des ports influencés de `newInfluentPort'`. Dans le cas contraire, la liste `newInfluentPort'.influencedPortList` sera complétée par chaque port influencé par `influencedPort'` à l'intérieur du sous-modèle du parent de `influencedPort'`. Ces structures seront ajoutées à la liste définitive `nonHierarchicalInfluentPortList`.

Enfin, Les relations de couplage internes des modèles inclus doivent être copiées dans la liste définitive `nonHierarchicalInfluentPortList`.

```
nonHierarchicalInfluentPortList CouplingTransformation (consideredCoupledModel)
for (all influentPort in consideredCoupledModel.InfluentPortListWithHierarchy) // upstream part of influence relation
    model = Retrieve parent with (influentPort.parentName)
    if (model != consideredCoupledModel && model != AtomicModel) // parent of port is not the coupled model
              includedModel = model // it is an included model
              for (all influentPortInIncludedModel in includedModel.influentPortList)
                        For (all influencedPortInIncludedModel in influentPortInIncludedModel.influencedPortList)
                        If (influencedPortInIncludedModel == influentPort)
                                  Create newInfluentPort
                                  newInfluentPort.name = influentPortInIncludedModel.name
                                  newInfluentPort.parentName = influentPortInIncludedModel.parentName
                                  newInfluentPort. influencedPortList = InfluentPort. influencedPortList
                                  add newInfluentPort in IntermediaryInfluentPortList
    else add influentPort in IntermediaryInfluentPortList // list for computation purpose only

for (all influentPort' in IntermediaryInfluentPortList) // downstream part of influence relation
    for (all influencedPort' in influentPort'. influencedPortList)
              influencedModel = Retrieve parent with (influencedPort'.parentName)
              if (influencedModel is non hierarchical || influencedModel is consideredCoupledModel)
                    // the coupling remain the same (no hierarchy)
                        newInfluentPort' = influentPort' // simple copy
              else // influencedModel is hierarchical
                        newInfluentPort'.name = influentPort'.name
                        newInfluentPort'.parentName = influentPort'.parentName
                        for (all influentPorIntIncludedModel in influencedModel. influentPortListWithoutHierarchy)
                                  if (influentPorIntIncludedModel.name == influencedPort'.name)
                                            for (all influencedPort in influentPorIntIncludedModel.influencedPortList)
                                                      newInfluentPort'.influencedPortList add influencedPort
add newInfluentPort' in nonHierarchicalInfluentPortList

for (all includedModel in consideredCoupledModel.includedModelWithHierarchyList)
    for (all InfluentPort'' in includedModel.InfluentPortListWithHierarchy)
              for (all InfluencedPort'' in InfluentPort.influencedPortList)
                        if (InfluentPort'' parent is AtomicModel && InfluencedPort'' parent is AtomicModel)
                                  if (InfluentPort'' is already in nonHierarchicalInfluentPortList)
                                  add InfluencedPort'' in InfluentPort''.influencedPortList
                                  else      add InfluentPort'' in nonHierarchicalInfluentPortList
```

Figure 51 - Fonction de transformation du couplage dans LSIS_DME

3.3.1.4 Illustration de l'algorithme de mise à plat de modèles G-DEVS

L'algorithme de mise à plat de la structure hiérarchique de modélisation et de simulation présenté dans le chapitre précèdent a été intégré à l'environnement LSIS_DME.

La Figure 52 illustre la mise à plat d'un modèle couplé hiérarchique DEVS M. Ce modèle est composé des modèles A et B provenant de la bibliothèque de modèles. Les fonctions de mise à plat permettent d'obtenir le modèle M' de la figure. Le modèle à simuler M' est donc composé de quatre modèles atomiques A1, B1, C1 et C2.

Les relations de couplage sont modifiées de la façon suivante : la relation de couplage définie par le port influenceur (Out2, B) est transformée en deux ports influenceurs (Out2, B1)

et (Out, C2). La relation de couplage définie par le port influenceur (Out1, B) est remplacée par (Out1, B1), par contre son ensemble d'influencés `InfluencedPortList` n'est pas modifié.

Enfin, la relation d'influence ayant pour port d'origine (Out, A) dont la liste `InfluencedPortList` contenait uniquement (In, B), est modifiée pour contenir (In, C1) et (In, B1).

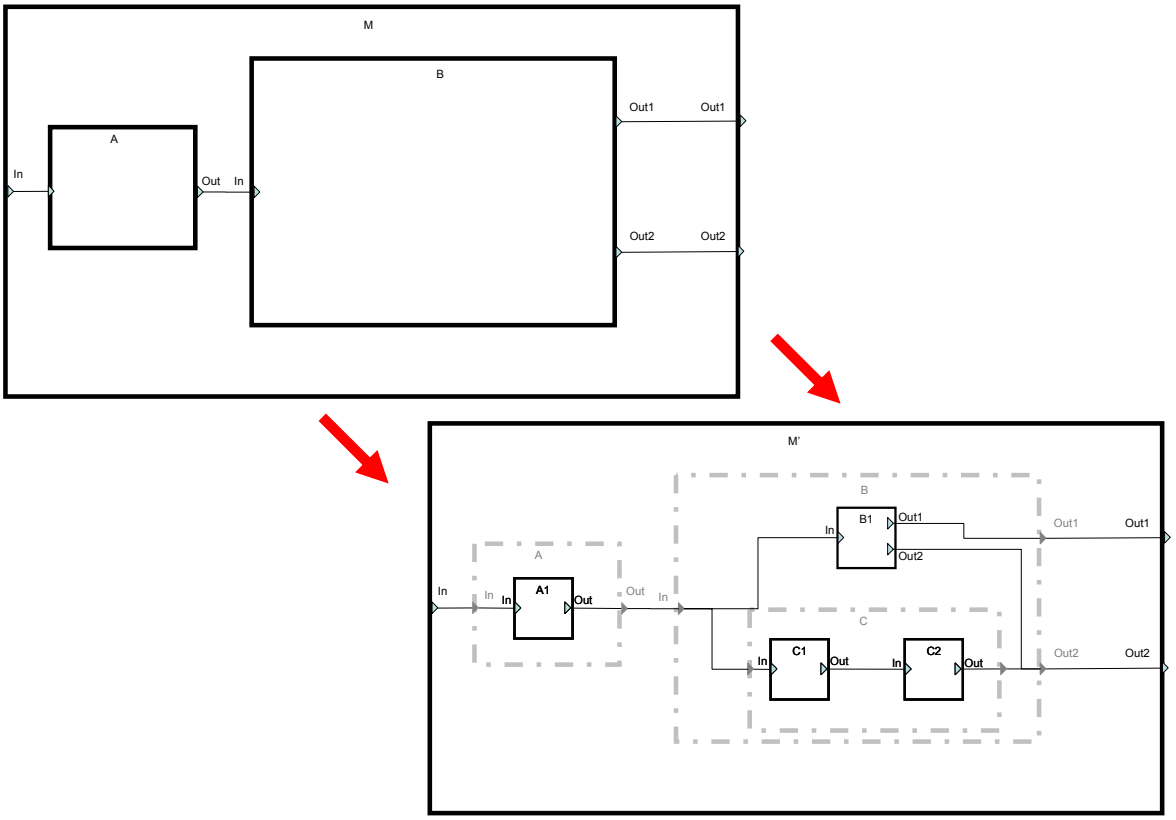


Figure 52 - Exemple de mise à plat de modèles

3.3.2 Algorithmes de simulation

Nous proposons une structure de simulation associée à la structure de modèles mise à plat. Elle comporte un processeur coordinateur et un ensemble de processeurs simulateurs comme cela est décrit dans le chapitre précèdent par la Figure 27 b). Nous allons, à présent, détailler les algorithmes des processeurs intervenant dans la simulation.

3.3.2.1 Diagramme de classe des simulateurs de LSIS_DME

La Figure 53 présente la partie du modèle de données relative à l'environnement de simulation. Cette représentation décrit les données de la classe Coordinateur et Simulateur, elle définit également la notion d'événement et met en évidence l'importance des listes dans cet environnement.

Les listes manipulent des messages échangés représentant les événements spécifiés dans les modèles. Les événements sont implémentés dans des objets nommés `Event` dont la structure décrit le contenu d'un événement du formalisme DEVS. Ils comportent un ensemble d'attributs : `Message Type` définit le type du message (*, x, y, d). L'attribut `receiver` définit

le destinataire, l'attribut `event time stamp` définit la date d'occurrence, l'attribut `concerned Port` définit le port destinataire. Enfin, l'attribut `Event Value` définit la valeur de l'événement. Cette dernière valeur est scalaire pour un modèle DEVS ou une liste de valeurs numériques pour représenter les trajectoires polynomiales des modèles GDEVS.

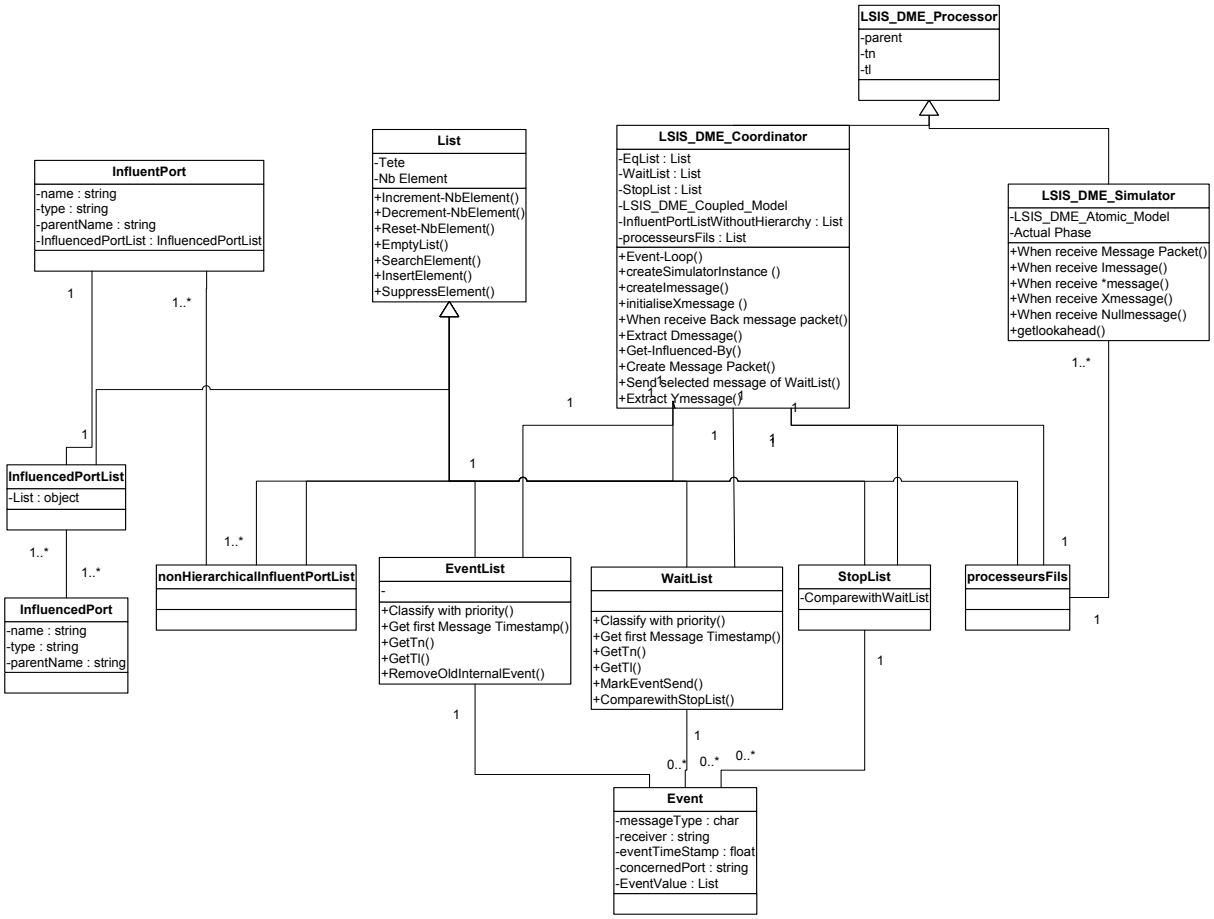


Figure 53 - Classes du Simulateur de modèles couplé DEVS dans LSIS_DME

3.3.2.2 Listes manipulées

L'exécution de l'environnement de simulation proposé est basée essentiellement sur la notion de liste et sur leur manipulation. Nous différencions principalement deux types de listes dans la classe `LSIS_DME_Coordinator`, dont nous présentons un descriptif ci-dessous.

Listes aux contenus statiques

La Liste **nonHierarchicalInfluentPortList** :

Cette liste est de dimension finie, qui reprend l'ensemble des relations d'influence définies dans le modèle. Notons que la classe de simulation ne considère que la structure de couplage non hiérarchique.

La Liste **processeurFils** :

Cette liste est de dimension finie. Elle contient l'ensemble des processeurs héritiers hiérarchiques du coordinateur considéré. Cette liste est créée à partir de la liste **includedModelWithoutHierarchy** des modèles inclus du modèle associé.

Listes aux contenus dynamiques

Les processeurs de simulation de modèles DEVS communiquent par échange de messages d'événements, qu'ils stockent dans des listes. Ces listes sont ensuite triées et actualisées ; enfin, les processeurs traitent les éléments provenant de ces listes. Ce type de listes est par conséquent de dimension variable.

Chacune de ces listes possède une fonction `Get first Message Timestamp()` lui permettant de communiquer sa valeur actuelle de tn. L'attribut tn est la date de prochain événement à traiter dans une liste d'événements. Dans le cas d'événements simultanés des règles de priorités sont mises en œuvre dans la fonction `Classify with priority()`. Il est néanmoins nécessaire de détailler les spécificités et les contextes d'utilisation de ces listes :

La Liste **Eq** : Cette liste est fréquemment nommée échéancier (EventList Scheduler). Elle contient l'ensemble des événements à traiter. Ces événements sont classés par date d'occurrence et, dans le cas d'événements simultanés, en fonction de règles de priorités définies dans une fonction similaire à la fonction Select définie par [ZEIGLER 00].

La Liste **WaitList** : Cette liste contient l'ensemble des événements transmis ou en cours de transmission. Les événements sélectionnés pour le traitement dans la liste Eq sont extraits de cette liste, insérés dans la liste WaitList et transmis au(x) successeur(s) concerné(s). Dans le cas d'événements simultanés à destination de différents simulateurs, seul le premier défini comme prioritaire est noté « *send* » et est transmis. Les autres seront transmis lorsque celui-ci aura été traité par les successeurs ; s'ils sont a destination d'un même simulateur il seront transmis en paquet.

La Liste **StopList** : Cette liste contient l'ensemble des événements reçus en retour des messages émis vers les simulateurs. Lorsque les simulateurs se sont acquittés de leur tâche de traitement des événements reçus, ils font parvenir en retour un message d'acquittement qui contient les informations du prochain événement local à traiter, et éventuellement un/des message(s) de sortie. Le contenu de la WaitList et de la StopList sont alors comparés afin d'éliminer les messages qui ont été traités.

Ces listes sont manipulées par les algorithmes du coordinateur. Dans le prochain point, nous présenterons en détails ces algorithmes.

3.3.2.3 Fonctions de la Classe Coordinateur

Nous présentons ici l'algorithme d'exécution du coordinateur. Il est composé de fonctions permettant la création d'instances de simulateurs ainsi, que la gestion de messages échangés au sein de la simulation.

Algorithme relatif à l'initialisation de la simulation

La Figure 54 présente les fonctions relatives à l'initialisation de la simulation.

La première fonction `createSimulatorInstance()` crée des instances de simulateurs pour chaque modèle existant dans l'ensemble des modèles atomiques de la structure "mise à plat" du modèle pris en paramètre.

La fonction `createImessage()` génère un message d'initialisation pour chaque simulateur. Ces messages sont ensuite insérés dans la liste Eq.

La fonction `initialiseXmessage()` génère des messages dans la liste Eq, en fonction des messages saisis par l'utilisateur lors de l'initialisation de la simulation. La génération tient compte des ports sur lesquels sont reçus ces événements, et des relations de couplage pour déterminer le modèle destinataire du message et par conséquent le simulateur associé.

```
DEVS Coordinator
Parent                // parent Distributed Root Coordinator
DEVS Simulator        // associated DEVS Simulator
Event                 // An Event is implemented into a message with the structure (Message
                      Type, receiver, event time stamp, concerned Port, Event Value)
tn                    // time of next event of the Event List
tl                    // time of last event
Eq                    // queue to store the containing of Event List sorted by occurrence date
                      and Select priority
WaitList              // queue to store the Event sent
StopList              // queue to store the Event treated
Value                 // value in YMessage and Xmessage is a list of value for GDEVS

    createSimulatorInstance (Atomic Model List)
    Create a Simulator instance for each Atomic Model

    createImessage ('i', - , t, - , - )

    for each Simulator of this coordinator do
              add Imessage ('i', Simulatior, t, -, - ) to Eq

    initialiseXmessage (User External EventList)

    for each External Event scheduled by User // trough the user interface "Add external
    Event"
       Get- Influenced-By (Coordinator, InfluencedPort)
       // consult external input coupling to get children influenced by the input

              for each Simulator influenced by input do
                          Add Xmessage ('x', Simulator, t, InfluencedPort, Value) to Eq
```

Figure 54 - Algorithme du coordinateur

Boucle de traitement principal de la simulation

La liste Eq contient un ensemble de messages (représentant chacun un événement) classés par dates. La boucle principale de traitement des événements est présentée Figure 55. Elle sélectionne un(des) message(s) dans Eq, transfère une copie de ce(s) message(s) en WaitList puis le(s) transmet aux simulateurs successeurs hiérarchiques directs. Lors de la sélection de messages, plusieurs situations peuvent être rencontrées :

La liste ne contient qu'un seul message de date tn :

> Une copie est conservée dans la WaitList et le message est alors transmis au successeur concerné.

La liste contient plusieurs messages de dates tn adressés au même simulateur :

La technique retenue pour l'envoi de messages de même date et à destination du même processeur est nommée « envoi en paquet ». Cette technique s'inspire des Parallel DEVS [CHOW 94]. En effet, les simulateurs possèdent une fonction `δcon()` d'interprétation d'un paquet de messages simultanés avec des règles de priorités spécifiques sur laquelle nous reviendrons dans les § suivants.

La liste contient plusieurs messages de dates tn adressés à des simulateurs différents :

Ces messages sont transférés dans la WaitList. Cette liste est ordonnée par date ; de plus dans le cas où elle contient des événements simultanés à destination de différents processeurs, une fonction de priorité (`Classify with priority()`) définit un ordre de priorité entre les processeurs et définit le classement de ce type de messages dans la WaitList. Le premier message ou paquet de messages de la Waitlist est donc transmis aux successeurs.

Après émission du message ou du paquet, le processeur actualise sa date et passe dans un état d'attente d'informations de retour, provenant de ses successeurs. Ces informations, contenues dans un message de retour, lui assureront l'acquittement du traitement du message émis par les processeurs successeurs.

La prise en considération du message retour est traitée dans le point suivant.

```
Event-Loop:
While (Eq != empty)
do Get first Message Timestamp in Eq
   Extract from Eq all messages with Timestamp t = First Message Timestamp t of Eq
   if (Multiple extracted Messages at timestamp t)
        then      if (Some of these Messages are addressee to the same receiver)
           then   Create Message Packet ('P', Simulator, t, InfluencedPort, messageList)
                  Add to Message Packet all messages intended to the same simulator
                  Add this Message Packet in WaitList
                      Classify with priority (extracted messages & Message Packet) in
                  WaitList // according to tie Breaking function

           else    Classify with priority (extracted messages) in WaitList // according to
                   tie Breaking function

   else Classify with priority extracted Message in Waitlist
                  // just one message with timestamp t
   Select first message of WaitList 'non send'
   Send this message of WaitList & mark it "send"
        tl = t
        tn = t first Event of the EventList
   Stop process: Wait for receive of Back message Packet from simulator
Endwhile
End Event-Loop
```

Figure 55 - Algorithme de boucle principale de traitement des événements par le coordinateur

Boucle de traitement des messages de retour provenant des simulateurs

La Figure 56 présente la fonction de traitement des messages en retour des simulateurs.

```
When receive Back message packet

   Extract Dmessage ('d', Simulator, t, - , Value)
   // inform when an Event as been processed by a hierarchical children
          add ('d', Simulator, t, - , Value) in StopList

          if (WaitList != StopList) // not all message have been reported of treatment
               then       if ( all message are already "send")
                     then    Wait for Back message Packet

                          else       Mark "send" first "non send" message of WaitList
                     Send this message of WaitList
                            Stop process: Wait for Back message Packet

                     tl = t
                     tn = t first Event of the EventList

          else     remove these messages from WaitList and StopList
                   Remove ('*', Simulator, next_internal_EventTime, -, -) from Eq
                   // old internal event associated to this Simulator
                   if (value != null)
                   then    extract next internal event ('*', Simulator, Value, - , - )
                                // value given by Dmessage
                           classify with priority ('*', Simulator, Value, - , - ) in Eq
                   tl = t
                   tn = t first Event of the EventList

   if (Back message packet contains Ymessage ('y', Simulator, t, Simulator_Port, Value))
   then    Get-Influenced-By (Simulator, Simulator_Port)
           // check for coupling to get children influenced by output
           for each children influenced by output
                       do Classify with priority Xmessage ('x', Simulator_Influenced, t,
                      Port_influenced, Value) in WaitList // according to priority definition

           if (output port of global Model are influenced)
                 // check for coupling to see if there is an external output event
           then    Print Output Event (Message)

                Send first 'non send' message of WaitList & mark it "send"
           tl = t
           tn = t first Event of the EventList
           Wait for Back message Packet

   Start again Event-Loop process

Get-Influenced-By (A, a) // According to the couple ((ModelA, PortA) (ModelB, PortB)) in
the Model definition List, the Output PortA influences the Input PortB

End DEVS Conservative-Local-Coordinator
```

Figure 56 - Algorithme de traitement des messages retour par le coordinateur

Lors de la réception d'un `Back_message_packet`, le coordinateur analyse son contenu afin d'en extraire les différents messages.

Tout message de retour contient au moins un *Dmessage* assurant l'acquittement du successeur du traitement du message considéré, ce message permettant d'actualiser le contenu de la StopList.

Dans le cas où l'état actuel du modèle successeur n'est pas passif, le message de retour contiendra de plus un **message*, qui permet de planifier dans Eq la prochaine transition interne du modèle associé au simulateur successeur, Notons que l'ancien **message* à destination du même successeur est supprimé à ce moment.

Le dernier message, pouvant être contenu dans le message de retour, est un éventuel *Ymessage*. S'il est présent, il est transformé en fonction des relations de couplage en *Xmessage* pour les processeurs influencés et ce *Xmessage* est directement inséré en WaitList. S'il n'y a pas d'autres messages en attente de transmission dans la WaitList, il sera transmis directement au successeur. L'algorithme Figure 56 illustre en détail ce traitement.

3.3.2.4 Fonctions de la classe simulateurs

Nous présentons dans la Figure 57, les fonctions comprises dans les objets simulateurs. Ces fonctions ont pour base les fonctions définies par [ZEIGLER 00], avec notamment la prise en considération de messages simultanés avec la fonction delta confluent.

```
DEVS Simulator
Parent            // parent coordinator
tl        // time of last event
tn        // time of next event
DEVS              // associated model with total state (s, e)

When receive Message Packet ('P', Simulator, t, InfluencedPort, messageList)
if (message in Message Packet is unique)
then
       When receive Imessage ('i', Simulator, t, - , - )
          tl = t - e
          tn = tl + ta(s)
          s= s0 // state initialisation
          add Dmessage ('d', Simulator, t, - , Value) to Back message packet
          send Back message packet to parent

       When receive *message ('*', Simulator, t, - , - ) at time t
          if (t= tn)
          then    y = DEVS.λ (s)
                  add Ymessage (y, t) to Back message packet
                  s = DEVS.δint (s)
                  tl = t
                  tn = tl + ta(s)
                  add Dmessage ('d', Simulator, t, - , Value) to Back message packet
                  send Back message packet to parent
          else error : bad synchronization

       When receive Xmessage ('x', Simulator, t, InfluencedPort, x)
          if (tl<= t <= tn)
          then    e = t - tl
                  s  = DEVS.δext (s, e, Xmessage)
                  tl = t
                  tn = tl + ta(s)
                  add Dmessage ('d', Simulator, t, - , Value) to Back message packet
                  send Back message packet to parent
          else error: bad synchronisation

else // messages in packet message are multiple
       When receive *message ('*', Simulator, t, - , - ) and Xmessage(s) ('x', Simulator,
   t, InfluencedPort, x) at time t
   if (t= tn)
   then    y = λ (s)
           add Ymessage (y, t) to Back message packet
           s = DEVS.δcon (s, Xmessage(s))
           tl = t
           tn = tl + ta(s)
           add Dmessage ('d', Simulator, t, - , Value) to Back message packet
           send Back message packet to parent
   else error : bad synchronization
end DEVS-Simulator
```

Figure 57 - Algorithme du simulateur

La particularité de l'algorithme du simulateur, par rapport à celui de DEVS basique, réside dans le format des messages reçus et émis. Comme il a été défini dans le coordinateur,

les messages peuvent être reçus en paquet. Le simulateur doit en extraire les événements contenus et les traiter en fonction de leurs natures et de leurs nombres. Le message de retour (`Back_message_packet`) est construit sur le même principe. Ce message contient le message d'acquittement (*Dmessage*) dont l'objectif est double : d'une part, il informe le coordinateur du traitement correct, au niveau simulateur, de l'événement reçu, d'autre part, il donne une nouvelle valeur de `tn` et `tl`. Il peut également contenir un éventuel *Ymessage* généré suite à une transition interne.

Nous rappelons dans la Figure 58, la description des fonctions des modèles invoquées dans les simulateurs. En particulier, si le message contient un événement interne et un (ou plusieurs) événement(s) externe(s), le simulateur appelle la fonction `δcon(s, Xmessage(s))` qui permet d'effectuer une seule transition pour plusieurs événements reçus simultanément. Cette fonction détermine une transition d'état à partir de règles basées sur les événements externes, leurs ports de réception et leurs valeurs, ainsi que sur les états du modèle.

```
DEVS.δext (s, e, Xmessage)    // this function produce a new state according to model specifi-
                      cation with the information of current state, condition on state vari-
                      able, Input event and elapse time in this current state.

DEVS.λ (s)            // this function produce an output according to model specification with
                      the information of current state, condition on state variable.

DEVS.δint (s)         // this function produce a new state according to model specification
                      with the information of current state, condition on state variable.

DEVS.δcon (s, set of Xmessage(s))    // confluent function; it is applied instead of δint when
                              inputs occur at the same time as an internal transition
```

Figure 58 - Fonction du modèle

3.3.3 Calcul de la complexité algorithmique

3.3.3.1 Rappel de complexité algorithmique

La complexité temporelle d'un algorithme est le nombre d'opérations élémentaires (affectations, comparaisons, opérations arithmétiques) effectuées par un algorithme. Ce nombre s'exprime en fonction de la taille *n* des données. On s'intéresse au coût exact quand cela est possible, mais également au **coût moyen** (on moyenne alors toutes les exécutions du programme sur des données de taille *n*), au cas le **plus favorable**, ou bien au **pire des cas**.

La complexité de l'algorithme est exprimée par $O(f(n))$[49], où f est classiquement une combinaison de polynômes, logarithmes ou exponentielles. Cette fonction reprend la notation mathématique classique, et signifie que le nombre d'opérations effectuées est borné par $cf(n)$, où c est une constante, lorsque n tend vers l'infini. Lorsque l'on s'intéresse à des problèmes polynomiaux, ce qui est majoritairement le cas en informatique, il existe souvent plusieurs

[49] La notation grand O, aussi appelée symbole de Landau, est un symbole utilisé en théorie de la complexité, en informatique, et en mathématiques pour décrire le comportement asymptotique des fonctions. Fondamentalement, elle indique avec quelle rapidité une fonction « augmente » ou « diminue » [CHIKINE 93].

algorithmes polynomiaux pour résoudre un problème donné. Dans ce cas, il faut effectuer une analyse de leur complexité algorithmique, afin de choisir l'algorithme le plus performant.

3.3.3.2 Complexité de l'algorithme de simulation hiérarchique

L'approche de simulation hiérarchique est basée sur l'association de coordinateurs à chaque nœud du modèle couplé. Cette représentation comporte donc autant de coordinateurs que de modèles couplés dans le modèle considéré. Chaque coordinateur possède un certain nombre de listes dans lesquelles il classe des événements, ainsi que des listes qui contiennent les informations relatives aux relations de couplage des modèles.

L'étude de la complexité peut être réduite (en simplifiant) à l'étude des listes dans lesquelles se trouvent les messages à manipuler et celles où l'on recherche l'information nécessaire pour router le message vers le processeur suivant. La Figure 59 illustre les listes associées au simulateur hiérarchique proposé par B.P. Zeigler.

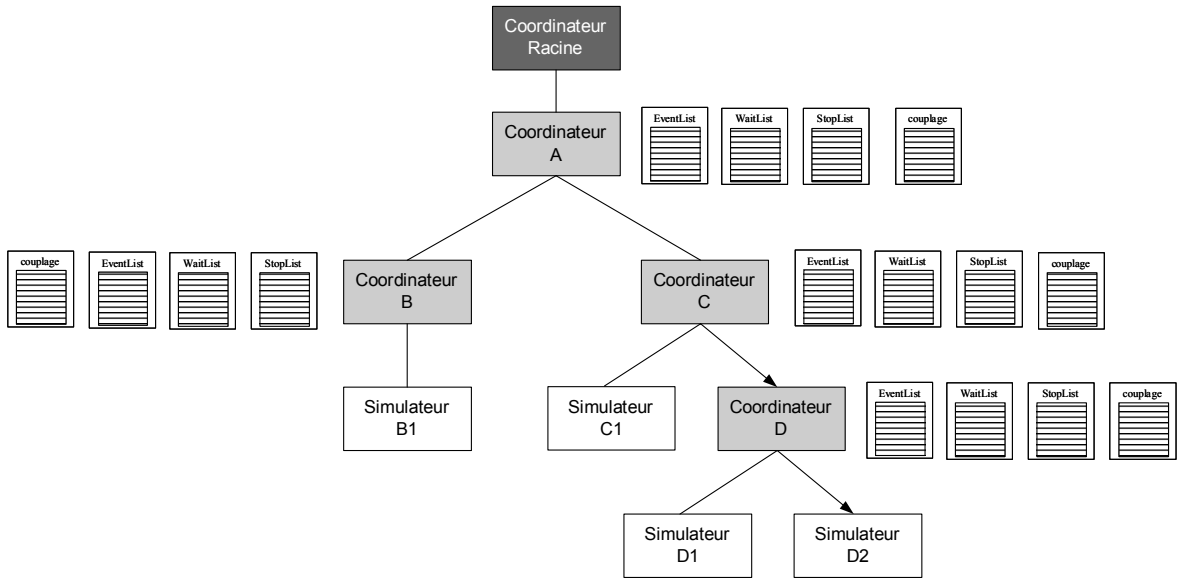


Figure 59 - Echéanciers de l'approche simulation hiérarchique

Considérons le cas d'un événement dans la EventList du Coordinateur A à destination du simulateur D1. Cet événement est le premier de l'échéancier, mais il est nécessaire de parcourir les premiers éléments du tableau afin de déterminer s'il n'existe pas d'autres événements de même date. Après cela, l'événement est supprimé de la EventList et classé dans le WaitList en fonction de sa date. Il est ensuite envoyé vers le processeur successeur concerné en fonction du résultat du parcours de la liste de couplage. L'événement est alors reçu par le processeur C, qui déclenche la suppression de l'événement de la EventList de ce processeur C, l'ajout dans sa WaitList, et la consultation de sa liste de couplage pour l'envoyer vers D. D déclenche à son tour la suppression de l'événement de sa EventList, l'ajoute dans sa WaitList et l'envoie, après parcours de la liste de couplage, vers D2. D2 traite le message et envoie un message d'acquittement à son supérieur hiérarchique D. D détermine, en fonction du contenu du message, s'il doit planifier un message pour d'autres processeurs influencés à son niveau. Pour cela, il utilise les définitions des relations de couplage. Il informe ensuite son supérieur

C de l'acquittement de son traitement et des éventuels messages de sorties à diffuser à un niveau supérieur. Enfin le processeur C itère la même opération que D en direction de A.

Le calcul de complexité dans le cas du simulateur hiérarchique peut être difficile si l'on s'attarde à décomposer l'intégralité de l'algorithme. Nous avons donc émis l'hypothèse simplificatrice consistant à nous concentrer sur les manipulations de listes qui sont effectuées par cet algorithme. En effet, la complexité des autres calculs de l'algorithme (de l'ordre de n, log(n) ou n.log(n)) est négligeable devant la manipulation des listes ou des tableaux (de l'ordre de n^2). La comparaison porte donc sur les algorithmes associés aux processeurs nommés coordinateurs (cf. processeurs A, B, C, D et Coordinateur racine Figure 59). Les processeurs simulateurs (cf. processeurs B1, C1, D1, D2) conservant une structure et des algorithmes quasi identiques dans les deux types de structures comparées, ils ne seront pas considérés dans la comparaison.

Les listes considérées sont La EventList, WaitList, StopList et la Liste de couplage. Nous définissons pour ces listes des tailles maximales.

La EventList du coordinateur de plus haut niveau est de taille **n**, cette valeur correspondant au nombre maximum de messages à traiter à un instant donné de la simulation. La WaitList, StopList sont de taille égale à la EventList.

La Liste de couplage est de taille **M** où M est le nombre de sous-modèles contenus dans le modèle considéré. Enfin, la représentation hiérarchique possède une structure d'arbre de profondeur **P**. A partir de ces postulats, nous proposons ici un calcul de la complexité moyenne pour le traitement des événements par la structure de simulation.

Malheureusement la profondeur **P** est difficilement calculable, elle est cependant bornée supérieurement par le nombre d'éléments M de la liste de couplage du modèle couplé le plus haut hiérarchiquement, tels que $P \leq M$. Nous pouvons ajouter que la complexité de la manipulation de l'arbre est linéaire en profondeur maximale car, pour la transmission d'un message, une seule branche est visitée. Nous expliquons ci-dessous pourquoi la complexité linéaire en taille de l'arbre doit être multipliée par les accès aux listes. Toutes les opérations agissent de la même manière, à savoir qu'elles appellent toutes une fonction récursive qui effectue ce qui suit :

Une série d'opérations sur le niveau hiérarchique sont effectuées, typiquement un ajout, une suppression et une recherche dans des listes de tailles fixées (dont on peut limiter le pire des cas par des constantes) est réalisée, puis un appel récursif à cette même fonction un étage plus bas, etc. Après avoir atteint la feuille de l'arbre la plus profonde (c'est-à-dire un processeur simulateur), ce processeur renvoie le résultat du traitement au coordinateur supérieur hiérarchique direct qui effectue la préparation des résultats à renvoyer à l'appel de ces mêmes fonctions au niveau supérieur. Ces opérations sont répétées avant d'avoir atteint le niveau coordinateur précédant le coordinateur racine.

Nous pouvons donc formaliser à chaque niveau de l'arbre de profondeur P, deux opérations de tri de liste de dimension n et une recherche sur une liste de dimension M. La relation de complexité est la suivante :

$$C1(P, M, n) = O(P\text{-}1) \times (O(\mathbf{M}) + O(\mathbf{n}^2) + O(\mathbf{n}^2)) \qquad (1)$$

A partir de la formule précédente, nous supposons les conditions suivantes : si la structure comporte un nombre important de modèles qui sont de plus très hiérarchisés, alors, P et M peuvent être considérés du même ordre de grandeur que n. Sous ces conditions, la simplification suivante peut être effectuée :

$$C1(n) = O(\mathbf{P} \times (\mathbf{M} + \mathbf{n}^2)) = O(\mathbf{n}^3) \qquad (2)$$

La complexité est, dans ces conditions, de l'ordre de $O(\mathbf{n}^3)$.

3.3.3.3 Complexité de l'algorithme de simulation non hiérarchique (avec mise à plat)

L'approche de simulation non hiérarchique est basée sur un seul coordinateur relié directement à l'ensemble des simulateurs. Cette représentation comporte donc un coordinateur et autant de simulateurs que de modèles atomiques. Nous présentons Figure 60 une structure de simulation non hiérarchique équivalente à la structure présentée Figure 59.

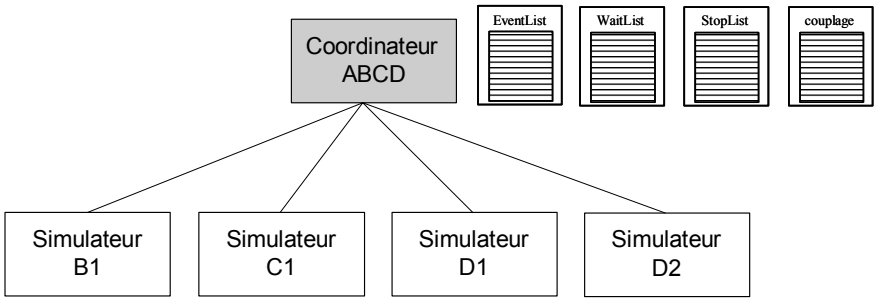


Figure 60 - Echéanciers de l'approche simulation non hiérarchique

Considérons le cas d'un événement dans la EventList du Coordinateur ABCD à destination du simulateur D1 :

Cet événement est le premier de l'échéancier, mais il est nécessaire de parcourir les premiers éléments du tableau, afin de déterminer s'il n'existe pas d'autres événements de même date. Après cela, l'événement est classé dans le WaitList en fonction de sa date. Il est ensuite envoyé vers le processeur successeur concerné D2. D2 traite le message et envoie un message d'acquittement à son supérieur hiérarchique ABCD. Enfin, ABCD détermine en fonction du contenu du message, s'il doit planifier un message pour d'autres processeurs influencés à sont niveau.

Nous supposons que deux opérations de tri de liste et une de recherche sont effectuées sur l'unique niveau de hiérarchie. Nous obtenons alors une complexité en $O(\mathbf{n'}^2)$, à partir des relations suivantes (3) et (4) :

$$C2(M', n') = (O(\mathbf{M'}) + O(\mathbf{n'}^2) + O(\mathbf{n'}^2)) \qquad (3)$$

A partir de la formule précédente, si nous supposons que le modèle considéré comporte autant de sous-modèle que de messages échangés (M' est alors du même ordre de grandeur que n'), alors les simplifications suivantes peuvent être effectués (4) :

$$C2(n') = (\mathbf{M'} + \mathbf{n'^2}) = O(\mathbf{n'^2}) \qquad (4)$$

3.3.3.4 Comparaison et limitations

La discussion s'axe ici autour de la comparaison de l'ordre de grandeur des données, en particulier P, n et n'. Les dimensions n et n' définissent respectivement la taille des listes d'événements de la structure hiérarchique et non hiérarchique. n' borne supérieurement n, en effet le nombre maximum d'événements dans les listes avec hiérarchie est toujours inférieur au nombre d'événement dans la liste unique de la structure non hiérarchique. La profondeur de la structure P détermine le nombre de listes à manipuler. La complexité de la structure hiérarchique croit plus rapidement devant la structure non hiérarchique car nous comparons ici une fonction linéaire et une fonction carrée.

En conclusion, si les listes sont du même ordre de grandeur et si la profondeur est une valeur non négligeable devant les deux premières, alors la structure de simulation hiérarchique est plus complexe et demande donc en théorie des temps de réponse également plus importants (5).

$$O(n'^2) < O(P \times n^2) \text{ / P non négligeable devant } n \approx n' \qquad (5)$$

A partir des informations issues de la source [GIRE 00] et dans le respect des hypothèses citées précédemment, nous pouvons approximer un temps de réponse pour les deux algorithmes. Pour un nombre de données n égal à 100, une machine effectuant 10^7 opérations/seconde (processeur 100 Mhz) aura des temps de réponse de l'ordre de 1 ms en version non hiérarchique et de l'ordre de 10 ms en version hiérarchique. Pour des listes de taille 10000, les réponses seront de l'ordre d'une dizaine de secondes pour une complexité quadratique et peuvent aller jusqu'à une heure pour les complexités cubiques. La Figure 61 présente la progression d'une fonction carrée en bleu et quadratique en rouge.

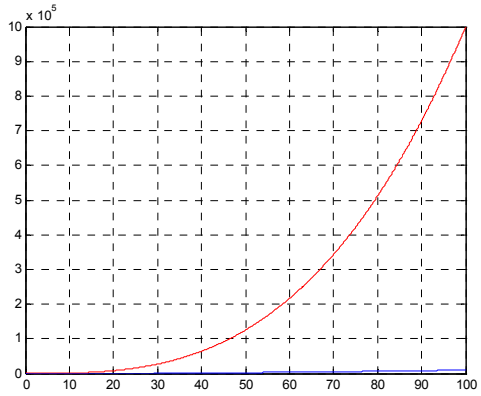


Figure 61 - Comparaison des complexités

Manifestement, il faut atteindre un compromis entre le nombre de processeurs coordinateurs et le nombre d'événements échangés, c'est-à-dire la recherche d'éléments dans un grand tableau ou plusieurs recherches d'éléments dans des listes de plus petites tailles. Cette comparaison n'est plus valide dans le cas où l'on travaillerait avec un très grand nombre de messages par rapport au nombre de modèles inclus et inversement.

Il est également important de retenir que les comparaisons de complexité occultent un certain nombre de points. En effet :

L'occupation mémoire n'est pas prise en compte. Par exemple, la comparaison de l'accès à un tableau de grande taille, ou à plusieurs listes de tailles moins importantes, est réalisée sans tenir compte du fait que les différentes listes sont rangées aléatoirement en mémoire. Ceci peut engendrer un surcoût en termes de temps d'accès aux données contenues.

Les problèmes d'entrées/sorties sont occultés. Par exemple, le traitement de l'acquisition des données et la transmission des informations entre les différentes fonctions des processeurs ne sont pas examinées.

En supposant que ces omissions ne faussent pas les calculs, le choix d'une structure mise à plat semble judicieux. Pour corroborer ce choix, nous présenterons des mesures de performance dans la suite de ce chapitre.

3.3.4 Mise en œuvre de la version distribuée de l'environnement G-DEVS/HLA

L'environnement de simulation distribuée spécifié dans le chapitre précèdent a été conçu et développé de la façon suivante.

La classe Coordinateur Racine local du simulateur séquentiel (*LSIS_DME_Coordinator*) présentée dans le § 3.2.2.1. a été dérivée pour faire appel aux méthodes définies dans le RTI (Figure 62). La classe fille, ainsi obtenue (*LSIS_DME_Coordinator_Distri*), implémente donc les méthodes abstraites de la classe *Federate_Ambassador*. Ces méthodes sont appelées par le RTI pour délivrer au fédéré des autorisations de traitement de données locales et des informations en provenance de la fédération.

En retour, cette classe du fédéré peut appeler les services proposés par le RTI pour s'enregistrer auprès de ce dernier, définir les interactions HLA auxquelles elle souhaite publier / souscrire, demander l'autorisation de traitement de messages, envoyer des messages vers la fédération et demander des informations relatives aux dates des classes des fédérés l'influençant.

Enfin la méthode *EventLoopDistrib()* (boucle de traitement des messages) implémente l'algorithme présenté dans le chapitre précèdent (cf. Figure 42) et elle remplace la méthode *Event-Loop()* utilisée dans le simulateur séquentiel. Cette méthode permet de communiquer avec le RTI, par l'appel des services RTI présentés dans le paragraphe précèdent, pour synchroniser le traitement des événements locaux et des événements provenant des autres coordinateurs.

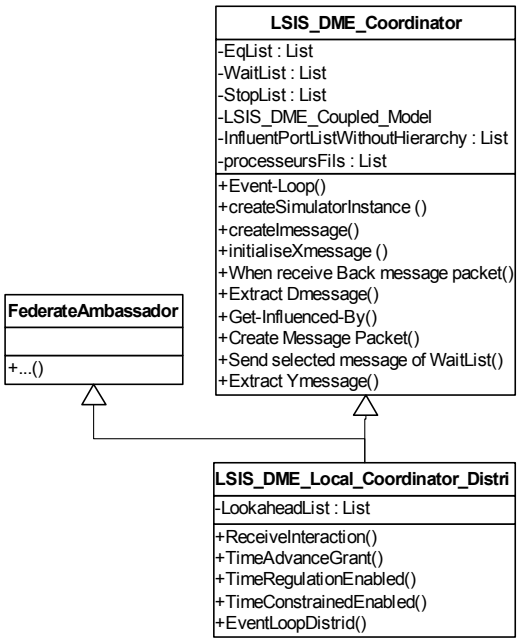


Figure 62 - Diagramme de classe de Local_Coord_Distrib

3.3.4.1 Considérations techniques de développement

La spécification du simulateur distribué présentée dans le chapitre précèdent a conduit au développement d'une maquette de simulation. Cette maquette implémente l'algorithme de communication des fédérés avec le RTI présenté dans le chapitre précédent. L'implémentation a été effectuée en langage Java avec les environnements de développement JBuilder [BORLAND 02] puis Eclipse [ECLIPSE 04].

Nous avons utilisé le RTI commercial pRTI 1516 de [PITCH 02]. Le choix de ce logiciel est lié au fait qu'il était à l'origine du développement du LSIS_DME, le seul à avoir été certifié par le DMSO pour le respect de la norme HLA version 1516 [DMSO 05]. De plus, Il est développé en Java, de même que l'environnement LSIS_DME, nous évitant ainsi les interfaçages avec d'autres langages de programmation. Enfin, nous proposons un exemple de fédération G-DEVS/HLA portant sur le modèle couplé « ComSysCom » dans la section suivante.

3.4 Illustrations de l'environnement de simulation

Nous proposons de modéliser des exemples simples dans l'objectif d'illustrer l'utilisation de l'environnement LSIS_DME dans sa version séquentielle pour le premier exemple et dans sa version distribuée pour le second. Ces exemples nous permettrons, de plus, d'analyser la trace résultante de sa simulation avec LSIS_DME.

3.4.1 Exemple Commande Système commandé

Nous présentons ici la modélisation et la simulation d'un modèle couplé DEVS d'un système commandé et de sa commande. Les modèles de ces systèmes interagissent pour produire un comportement autonome : la commande envoie des ordres au système commandé, qui répond en modélisant et en simulant des capteurs d'accomplissement de tâches. Il est possible

de donner certains ordres depuis l'extérieur, c'est-à-dire en entrée du système de commande ; il est en effet parfois nécessaire de définir un accès externe aux commandes. Ce système fonctionne donc de façon autonome en échangeant des messages, mais il peut être perturbé par un événement externe reçu sur le port d'entrée nommé « Com ». Ce message d'interruption permet, dans ce cas, de réinitialiser le modèle de système commandé qui se fige alors dans un état *puit*.

Cet exemple est illustré ci-dessous dans la représentation textuelle proposée par [ZEIGLER 00] ainsi qu'en notation graphique selon [SONG 95] dans la Figure 63.

Le modèle couplé System (S) est défini par:

$Iports_S$ = {Com} and $Oports_S$ = {Ø};
D = {Command, Commanded Syst.};
EIC = {((System, Com), (Commanded Syst., Com));
IC = {((Command, E), (Commanded Syst., E));
((Commanded Syst., O), (Command, O))}

Le modèle atomique Command est défini par:

S = {S1, S2, S3, S4, S5}; XM = {Com, O};
YM = {E}; s° = S1

Le modèle atomique Commanded Syst. est défini par:

S = {S1, S2, S3, S4}; XM = {E};
YM = {O}; s° = S1

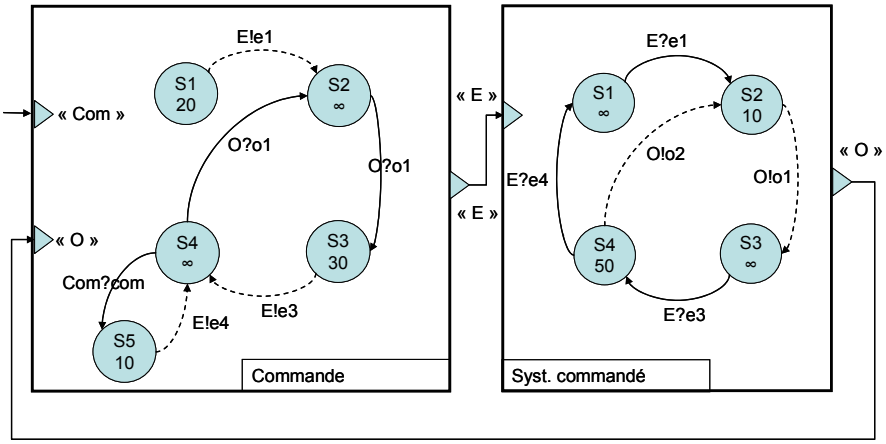


Figure 63 - Modèle Commande Syst Commande

Les modèles atomiques sont créés graphiquement puis stockés en librairie (la Figure 47 présente une version du modèle atomique Commande crée dans LSIS_DME avec une variable d'état supplémentaire notée *cond*). Ces éléments sont ensuite utilisés en terme de composants de base pour définir un modèle couplé. La construction de ce modèle couplé, dans LSIS_DME, a servi précédemment d'illustration au § 3.1.2 dans la Figure 48. Dans cette fi-

gure, un modèle couplé nommé « ComSysCom » est créé en utilisant les modèles atomiques « Commande » et « SystCommande » de la librairie nommée « Zach ».

3.4.1.1 Saisie des événements d'entrée du modèle

Lors du lancement de la simulation, une fenêtre permet de planifier les événements d'entrée du modèle pour sa simulation. Pour cela, nous définissons un port d'entrée, une valeur et une date d'occurrence pour chaque événement.

Le modèle couplé ne comporte qu'un seul port d'entrée et l'ensemble des valeurs recevables est réduit à une seule valeur. La Figure 64 illustre la saisie d'un événement externe de valeur « com » planifié sur le port « Com » à la date 65, dans le cadre de l'exemple considéré.

La fenêtre de lancement nous permet également de définir une date de terminaison maximale à la date 150.

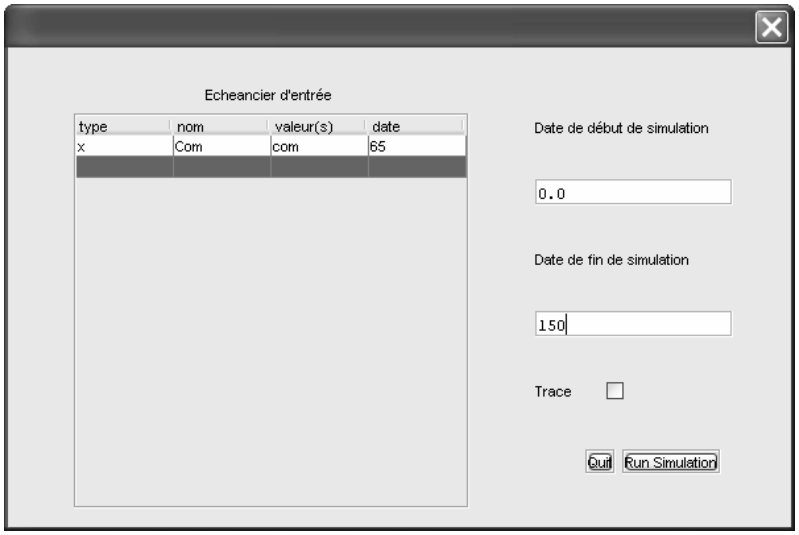


Figure 64 - Fenêtre de planification des événements d'entrée

3.4.1.2 Analyse des résultats de simulation

Apres terminaison de la simulation, l'environnement affiche les résultats de la simulation dans un tableau détaillant la trace de la simulation (Figure 65). Ce tableau propose des listes donnant le détail des événements manipulés au cours de la simulation. Ces listes effectuent la distinction entre les événements traités, non traités et les événements générés en sortie.

Dans l'exemple considéré, il est possible de vérifier que l'événement externe de valeur « com » planifié sur le port « Com » à la date 65 à été effectivement traité. En conséquence, un événement e4 a été émis et il est ainsi possible de savoir que la phase S1 du modèle Syst.Com a été atteinte une nouvelle fois à la date 75.

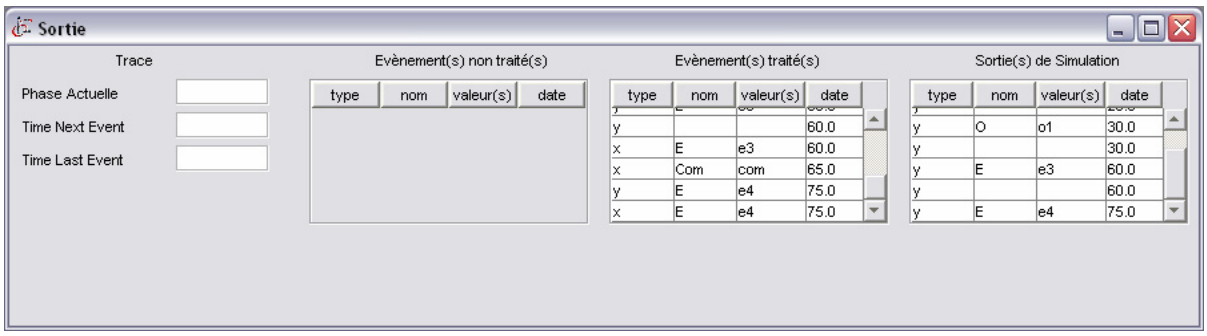


Figure 65 - Fenêtre d'affichage de la trace de la simulation

3.4.2 Exemple de Modèle couplé distribué G-DEVS/HLA

Nous avons illustré dans [ZACHAREWICZ 05c] le fonctionnement de l'algorithme de communication sur un exemple. Nous considérons à nouveau dans ce paragraphe le modèle couplé « System » G-DEVS que nous avons introduit précédemment, illustré Figure 63. Nous proposons à présent une version distribuée où chaque sous-modèle et son simulateur sont insérés dans un fédéré. La Figure 66 illustre cette version.

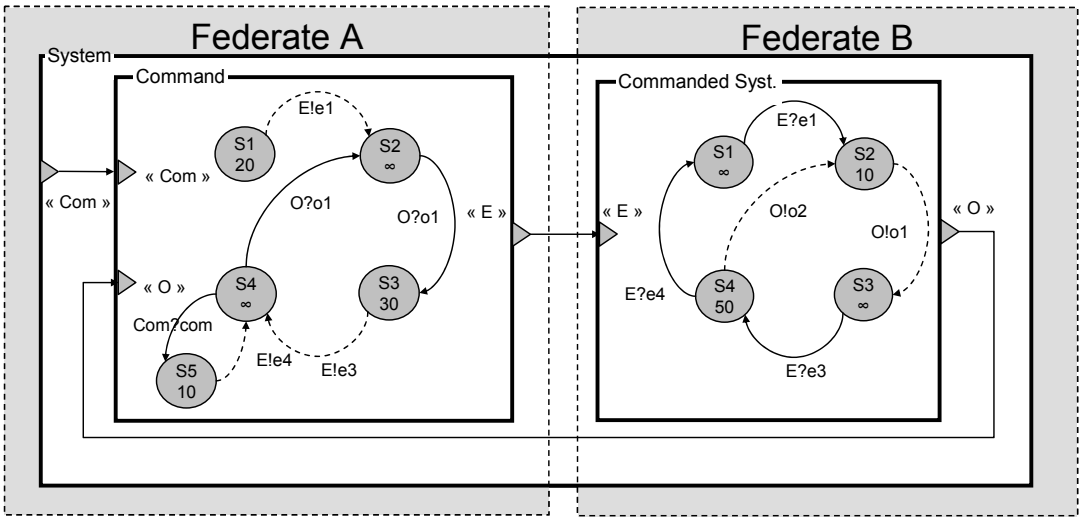


Figure 66 - Exemple de modèles couplés Distribués

Plus en détails, nous souhaitons exécuter les modèles « Command » et « Commanded Syst. » sur deux ordinateurs distants. Pour cela, l'environnement propose de les intégrer dans une simulation distribuée compatible HLA, en créant pour cela une fédération et en remplissant le FOM associé.

La fédération contient ainsi deux fédérés associés chacun à un modèle couplé : le fédéré A (FA) contient le modèle et le LCS G-DEVS de « Command », et le fédéré B (FB) contient le modèle et le LCS G-DEVS de « Commanded Syst ».

Afin de suivre les recommandations du FEDEP, nous proposons l'élaboration d'un FOM. Ce FOM contient les objets et les classes d'interactions partagées dans la fédération. Dans le cas considéré, les attributs des deux modèles sont intégrés en objets et les relations de couplage sont intégrées en interactions. La fédération contient ainsi trois interactions « E » et « O » et « Com », qui sont représentées dans la table Figure 67. FA souscrit aux interactions « O » et « Com » et publie sur l'interaction « E », tandis que FB souscrit à l'interaction « E »

et publie sur l'interaction « O ». L'interaction « Com » issue du SOM de FA pourra être publiée par un autre fédéré se joignant ultérieurement à l'exécution de la fédération.

Interaction Class Structure Table			
HLAinteractionRoot (N)	CouplingRelation (PS)	ExternalInputCoupling (S)	EIC "Com" Port of Model System -- "Com" Port of Model Command
		InternalCoupling (PS)	IC "E" Port of Model System -- "E" Port of Model Commanded Syst.
			IC "O" Port of Model Commanded Syst. -- "O" Port of Model Command

Figure 67 - Tables du FOM du modèle Commande--Syst. Commande

Lors de leurs inscriptions auprès de la fédération, les fédérés définissent leurs valeurs de Lookahead. Dans le cas étudié, FA et FB possèdent tous deux un Lookahead de 10 unités de temps, ce qui signifie qu'il devra s'écouler au moins 10 unités de temps entre la réception d'un message et l'émission d'un autre message résultant de ce message reçu. Les fédérés sont à la fois régulateurs de temps et contraints de temps et peuvent donc envoyer et recevoir des messages datés. Dans le programme de chaque fédéré, l'environnement intègre la boucle algorithmique présentée dans le § précédent, permettant de choisir l'événement local suivant du fédéré et d'interroger le RTI sur l'autorisation à le traiter.

La Figure 68 illustre la communication, entre FA et FB de la Figure 66 au travers du RTI. Cet exemple montre l'intérêt d'utiliser le service *queryLITS*() comme proposé dans l'algorithme Figure 42 du chapitre 2. Les valeurs chiffrées représentent les unités de temps logique, correspondant dans le schéma aux temps logiques des fédérés ou à la date des messages envoyés. Les arcs représentent des relations causales. Les comparaisons temporelles de l'exemple sont définies par référence à un temps « d'horloge murale », comme défini dans [FUJIMOTO 98], qui mesure le temps pendant l'exécution de la simulation. Par exemple, dans la Figure 68, l'ordre alphabétique de lettres entourées représente un ordre dans le temps d'horloge murale.

Nous nous concentrons, à présent, sur une succession de précédences causales, représentées Figure 68 par les arcs pointillés et les lettres entourées par les symboles ⓨ. Cet exemple illustre une demande *queryLITS*() de FA auprès du RTI (noté ⓒ), intervenue après que celui-ci ait reçu un *queryLITS*() (noté ⓐ) et un *NMRA*(∞) (noté ⓒ) provenant de FB.

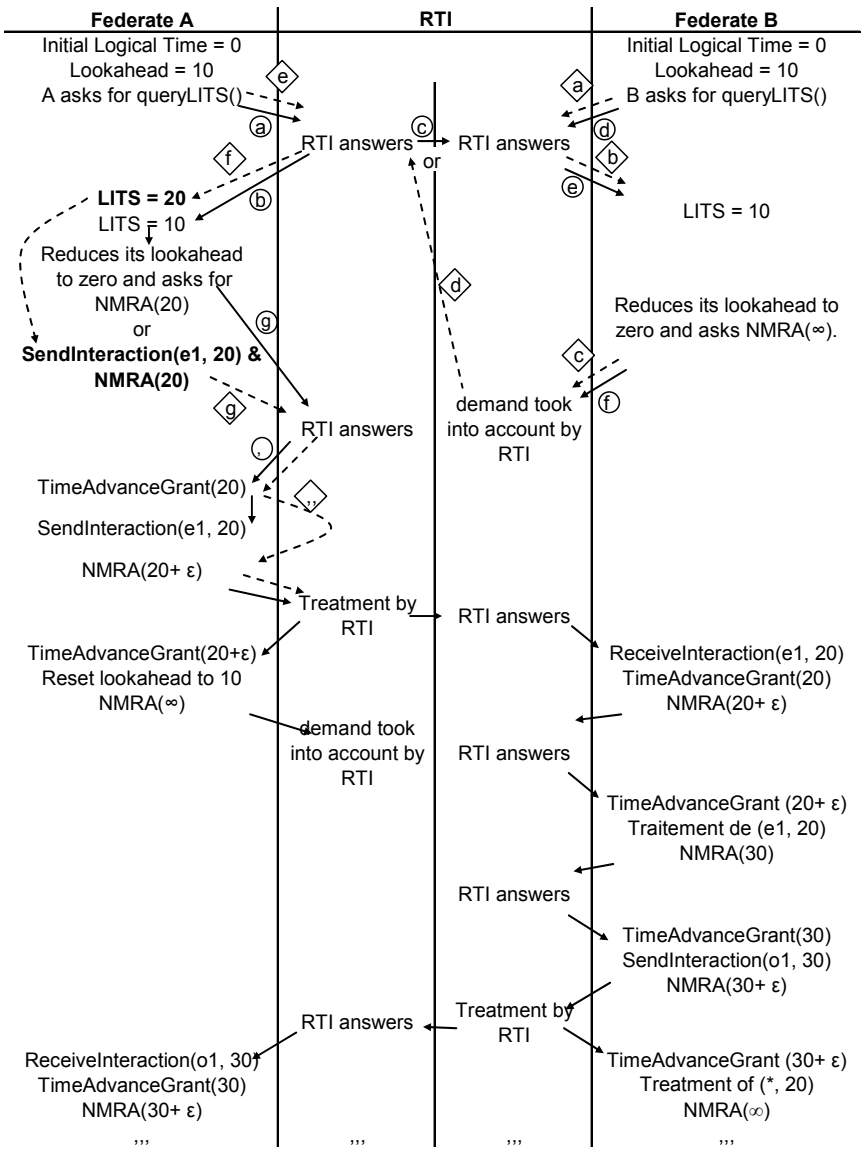


Figure 68 - Communication entre les fédérés au travers du RTI.

Dans cet exemple, FB a informé avec *NMRA*(∞) (noté ◇c) le RTI, qu'il n'émettrait pas de message de sortie sur « O » à la date 0. Concrètement, son état actuel est passif et aucune transition interne n'y est donc associée. Ainsi, le message de sortie suivant de FB doit être une conséquence (indirecte) d'un message reçu sur le port « E ». En outre, les messages reçus de FB résultent de FA, et FA a informé le RTI que son temps logique est 0. Son prochain message émis sera donc daté au minimum 10 (à cause de son Lookahead). En résultat, le message sortant suivant de FB est borné inférieurement par la relation (7).

FB LITS (Min (FA Lookahead (=10), RTI Message time for FB)) + FB Lookahead (=10) = 20 (7)

D'autre part, le LITS de FA est calculé par le RTI en fonction de la souscription de FA à l'interaction « O » publiée par FB. En réponse à *queryLITS*() (noté ◇e), le RTI calcule donc le LITS de FA égal à la date de message de sortie minimum de FB, c'est-à-dire 20 (7). Le fédéré FA peut donc émettre immédiatement son message de sortie planifié à la date 20 (noté ◇g) et préserver un Lookahead de 10 unités de temps, car nous avons défini une priorité de l'événement interne en cas de la simultanéité.

Le fédéré FA conserve dans ce cas un Lookahead non nul et libère ainsi de sa contrainte son fédéré FB influencé pendant une période au moins égale à son Lookahead et améliore ainsi le parallélisme de la simulation. Notons que cette situation est très désirable, mais n'est pas systématique dans les simulations. Parfois, en fonction de la progression du temps d'horloge murale des différents fédérés, certains d'entre eux peuvent être amenés à réduire leur Lookahead à 0 comme décrit le cas représenté par les lettres encerclées et les arcs pleins dans l'illustration Figure 68.

4 Performances

4.1 Performances des environnements DEVSCluster & DDEVSim ++

[KIM 00] ont conduit des expériences de simulation en utilisant leur algorithme de simulation non hiérarchique DEVSCluster et leur algorithme de simulation hiérarchique précédent, DDEVSim ++. La configuration de leur plate-forme de simulation était quatre Pentium III sous Windows NT interconnectés par Ethernet à 10 Mbps. Ils ont mis en oeuvre leur système de simulation en Visual C++ 6.0 avec une GUI en Java. L'étude portait sur la comparaison des versions séquentielles de DEVSCluster et DDEVSim++. Le résultat de cette comparaison de performance affiche que DEVSCluster s'exécute deux fois plus rapidement que DDEVSim++. Ce résultat provient notamment de la différence de méthode de traitement des événements car DEVSCluster manipule les événements par des appels de méthode d'objets, tandis que DDEVSIM ++ manipule les événements par passage de message explicite. La performance de la forme non hiérarchique provient également du fait que le nombre de *Xmessages* et *-messages générés et gérés par DEVSCluster est réduit par un facteur, par exemple d'un quart, pour une structure comportant quatre nœuds. Par extrapolation, ce facteur semble suivre une tendance quadratique.

L'approche de [GLINSKY 05] propose également une mesure de performance, dont nous rappelons les résultats suivants. L'approche plate a permis la simulation d'un modèle contenant 13 × 9 sous-modèles (97 composants). Ces modèles comportent deux ports d'entrée et deux ports de sortie, les interconnections sont élaborées dans le but de générer un nombre important de transitions et donc une surcharge d'échange d'informations. Malgré cela, les temps de réponse minimaux dans le pire des cas de ce modèle ont respecté toutes les dates limites imposées. En utilisant l'approche plate, on remarque d'abord une dégradation de performance à l'exécution d'un modèle 14 × 9 (105 composants). Au contraire, même en simulant un plus petit modèle 7 × 9, l'approche hiérarchique avait un pourcentage de succès inférieur (87 %) et a montré des temps de réponse moins rapides. Ils montrent que dans de plus grands modèles, la différence de temps de réponse devient considérable, de part la surcharge encourue par la simulation de modèles « plus profonds ». Leur analyse a montré des résultats semblables quand des modèles avec un nombre d'éléments variables sont exécutés. De façon similaire à l'illustration précédente, le coordinateur plat simule, plus efficacement, des modè-

les plus larges. DEVStone fait apparaître que les techniques de simulation hiérarchiques sont plus adaptées à la simulation de modèles peu hiérarchiques, même si les modèles possèdent une structure complexe. En général, [GLINSKY 05] conclut que la technique « plate » surpasse la version hiérarchique, réduisant la surcharge encourue jusqu'à 50%, fournissant donc des temps de réponse améliorés et un meilleur pourcentage de succès dans l'exécution. Ainsi, l'utilisation de l'approche non-hiérarchique permet la simulation de plus grands modèles avec des meilleurs résultats d'exécution. Ces résultats sont une conséquence du nombre réduit de messages échangés dans le mécanisme de simulation plat.

En résumé, les résultats des tests précédents démontrent que la technique de simulation mise à plat peut améliorer l'efficacité du simulateur dans certains cas, particulièrement quand la structure modèle est très grande ou complexe. Indépendamment de la taille et de la complexité des modèles, le simulateur plat surpasse en moyenne, par ses performances la version hiérarchique. En général, les diagrammes illustrent que la surcharge encourue par le simulateur plat est réduite jusqu'à environ 55 % par rapport à l'approche hiérarchique.

4.2 Mesures de performance réalisées sur le simulateur G-DEVS « mis à plat »

Nous avons réalisé des tests sur une maquette utilisant le simulateur non hiérarchique de LSIS_DME et le moteur de simulation DEVS développé précédemment utilisant une structure hiérarchique. La configuration de notre plate-forme de simulation est un Pentium III 2.4 GHz avec 512 Mo de RAM sous Windows XP. Les deux simulateurs sont implémentés en Java.

Les mesures sont réalisées à l'aide de l'outils JRat[50] [DROST 01] permettant d'effectuer des mesures de performances dans des programmes Java en incluant des classes spécifiques qui ont pour fonction d'effectuer des mesures quantitatives et qualitatives sur l'exécution du code. Une fenêtre de résultats proposés par cet outil est présentée dans la Figure 72.

4.2.1 Exemples mis en œuvre

Nous avons effectué une série d'études de comparaisons sur des modèles couplés dont les caractéristiques permettent d'exprimer un ensemble représentatif du spectre des modèles concevables. Nous présentons ici l'étude pour deux modèles couplés « cas d'école ».

Le premier modèle couplé A (cf. Figure 69) est un modèle simple, faiblement interconnecté avec les entrées et sorties du modèle considéré et des sous-modèles. Le modèle couplé B (cf. Figure 70) possède un nombre plus important d'interconnections entre les sous-modèles couplés. Ces deux modèles « cas d'école » ne proviennent pas de modélisations de cas réels, ils permettent principalement d'analyser l'influence de l'ajout d'un composant Coordinateur à chaque encapsulation d'un modèle atomique sur les performances de simulation.

[50] JRat est un Toolkit d'Analyse de Temps d'exécution de code Java. Son but est de permettre aux développeurs de mieux comprendre le comportement pendant l'exécution de leurs programmes Java. Le terme comportement inclut l'étude de la performance et la répartition des charges de chaque sous programmes.

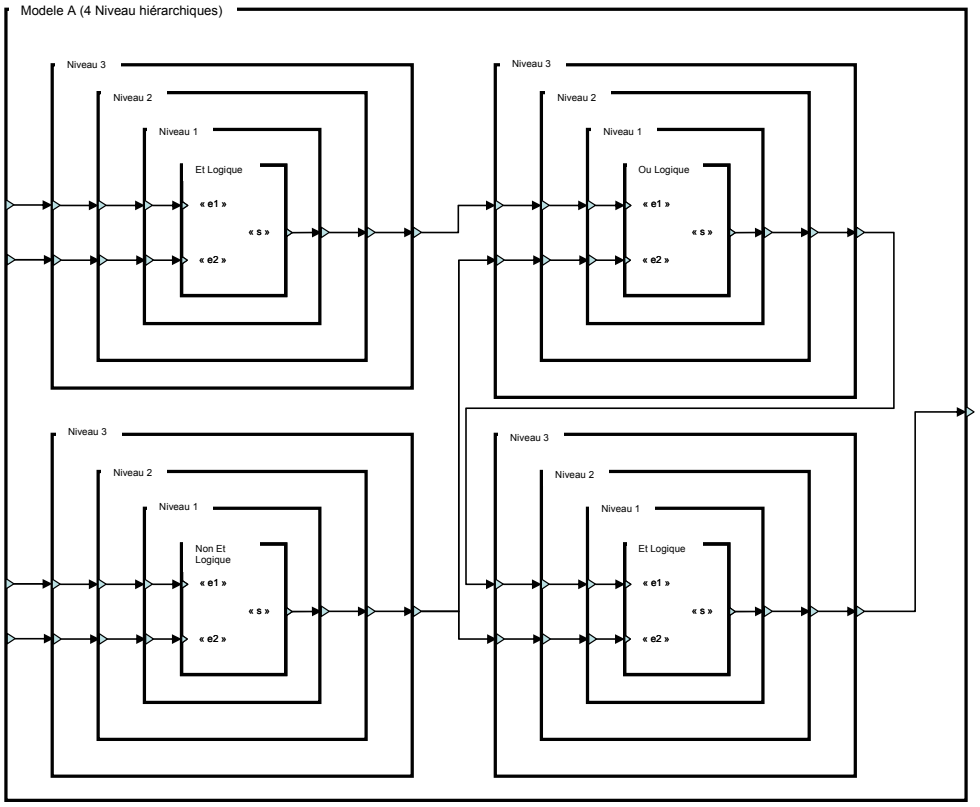


Figure 69 - Modèle A hiérarchique

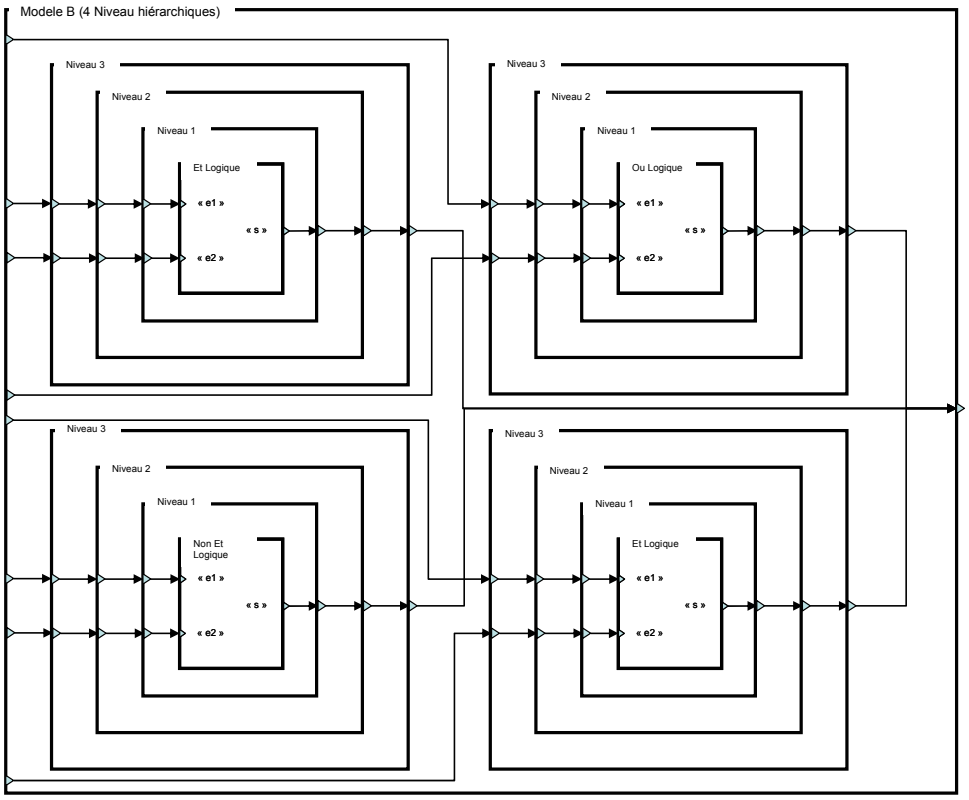
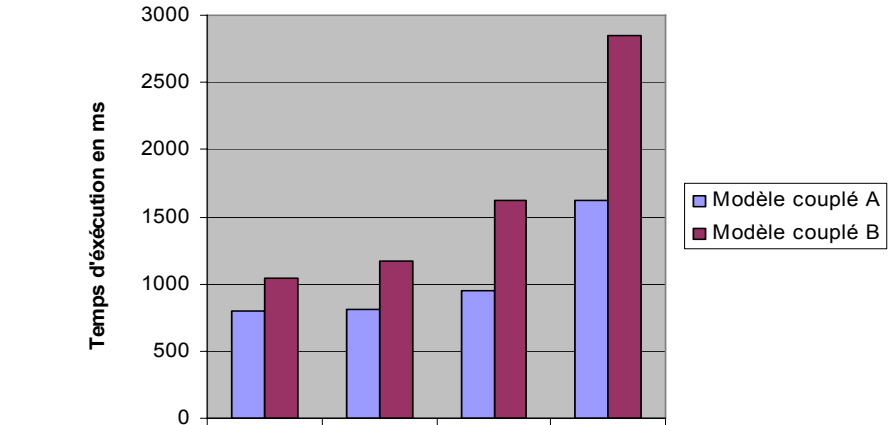


Figure 70 - Modèle B hiérarchique

Les modèles mis à plat équivalant à A et B contiennent les 4 modèles atomiques G-DEVS, de composants logiques, reliés sur un même et unique plan hiérarchique. Nous avons défini pour chacun de ces types de modèles une imbrication dans des structures plus ou moins

hiérarchiques (4 niveaux dans les Figure 69 et Figure 70). Les niveaux hiérarchiques retenus vont de un niveau à douze niveaux pour chacun des modèles. Pour chacune de ces structures nous avons exécuté un nombre de réplication significatif afin de les comparer le plus objectivement possible. La simulation a utilisée 100 événements planifiés en entrée.

La Figure 71 rapporte le temps d'exécution en milliseconde en fonction du nombre de niveaux hiérarchique (et donc du nombre de composants Coordinateurs ajoutés) pour chacune des structures. Elle fait apparaître la croissance du surcoût en temps d'exécution entre une structure hiérarchique et non hiérarchique lorsque la hiérarchie augmente au sein des modèles.



	1 niveau	2 niveaux	6 niveaux	12 niveaux
Modèle couplé A	797	812	945	1625
Modèle couplé B	1047	1172	1625	2844

Figure 71 - Comparaison des performances « mise à plat » et hiérarchique

4.2.2 Discussion

Il faut retenir que les tests effectués sont réalisés sur des simulateurs utilisant des modèles peu complexes en termes de nombre de modèles inclus. De plus, nous n'avons pas défini de profondeurs hiérarchiques trop importantes. Malgré cela, cette étude conclue que la version non hiérarchique de la structure de simulation est plus performante pour ces types de modèles. Ces résultats convergent avec les expériences effectuées par [GLINSKY 05] et [KIM 00] et permettent d'étayer les hypothèses théoriques et le calcul de complexité algorithmique énoncés précédemment.

Par ailleurs, les mesures de performance ont fait apparaître certaines limites liées au développement de LSIS_DME. Indépendamment de la partie graphique, nous avons effectué des séries de tests permettant de détailler les durées d'exécutions des différentes sous-fonctions de l'environnement. Il apparaît notamment une durée d'exécution importante liée à la lecture des modèles sauvegardés en code sérialisé Java, cette proportion peut atteindre quatre vingt pourcents du temps d'exécution pour des simulations courtes et un faible nombre d'événements traités. Cette situation est rencontrée dans la simulation de modèle simples types « cas d'école ». L'implémentation du simulateur étant réalisée dans le cadre d'un projet universitaire, nous supposons qu'elle pourrait être optimisée dans le cadre d'un développe-

ment professionnel. La Figure 72 illustre sur une réplication ce propos, où la classe *getLib*() récupère la librairie, *lance_Simu()* exécute la simulation et *New_Class*() instancie le coordinateur unique et les simulateurs.

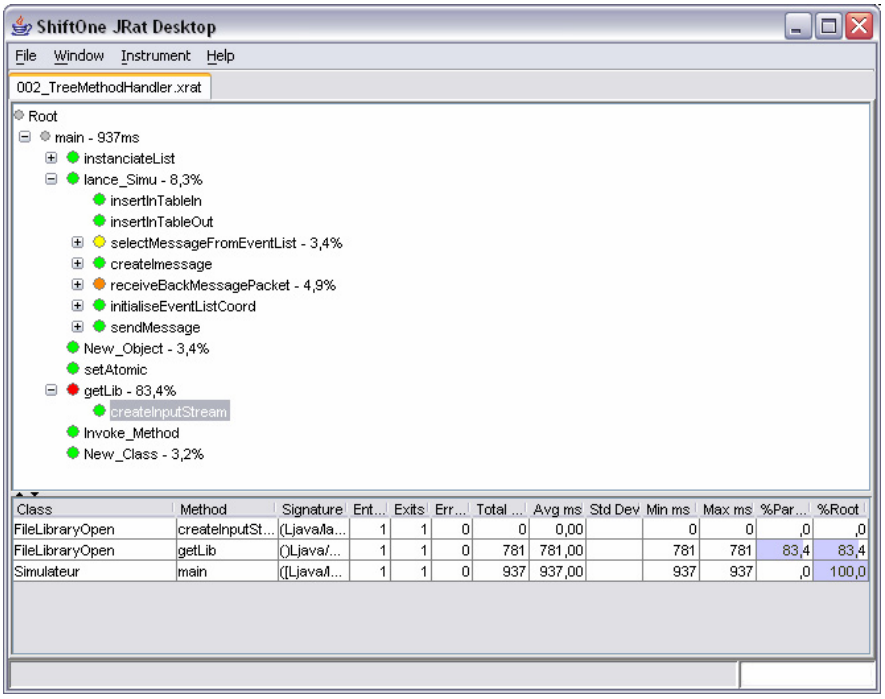


Figure 72 - Arborescence de sortie des performances avec JRat

4.2.3 Amélioration de la gestion de listes

Une autre amélioration significative pourrait être envisagée, en effet, la lacune communément admise de la structure de simulation à plat provient de la gestion de liste comportant un nombre important de messages.

Le problème d'un grand échéancier (insertion et classement des messages) peut être amélioré grâce à des heuristiques paramétrables en fonction des durées de vie des modèles. Un exemple d'une telle heuristique à ce niveau peut être donné par l'heuristique qui consistait à définir un échéancier avec un futur proche (date limite paramétrable) et un deuxième échéancier représentant un futur plus lointain proposée par [GIAMBIASI 79] et [MIARA 78]. Dans ce cas, le nombre d'échéanciers et leurs gestions dépendent des paramètres du modèle simulé et des retards décrits dans celui-ci et non pas de la structure décrite par l'utilisateur. De plus, il est évident que le nombre de messages échangés dans ce genre d'approche n'est pas augmenté, et que les événements sont également limités par échéancier.

4.3 Mesure de performance de simulations G-DEVS / HLA

Nous avons mis en œuvre les algorithmes des Figure 42, 43 et 45 du chapitre 2 sur deux ordinateurs Pentium 4 cadencés à 2.4 GHz, avec 256 Mo de RAM, ayant pour système d'exploitation Windows XP et étant interconnectés par un réseau local de 10 MbPs. Ces résultats ont été présentés dans [ZACHAREWICZ 06a].

4.3.1 Exemples mis en œuvre

Dans les essais présentés ici, chaque fédéré contenait un modèle atomique G-DEVS simple. La Figure 73 illustre les modèles à deux phases MA et MB mis en oeuvre. Le premier modèle est initialement dans un état transitoire permettant d'émettre un événement de sortie après écoulement de la durée de vie. Le second modèle est initialement dans un état passif attendant la réception d'un événement d'entrée.

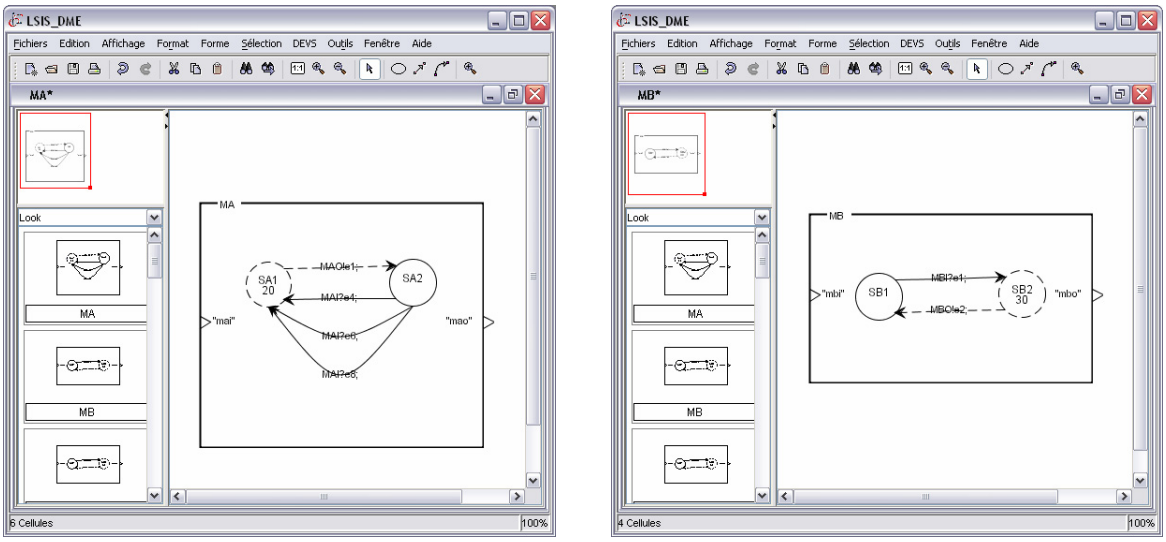


Figure 73 - Modèles atomiques G-DEVS

Chaque fédéré publiait/souscrivait aux interactions HLA permettant de définir le couplage d'un modèle couplé G-DEVS distribué, représenté par une fédération composée d'une « chaîne fermée » de fédérés modèles couplés. Le modèle séquentiel équivalent à un modèle couplé distribué à 8 fédérés est présenté Figure 74. Nous retrouvons dans ce modèle couplé, les modèles atomiques MA et MB introduit ci-dessus. Les autres modèles (MC … MH) possèdent un comportement similaire à celui de MB, leurs états initiaux sont passifs et attendent donc la réception d'un événement d'entrée.

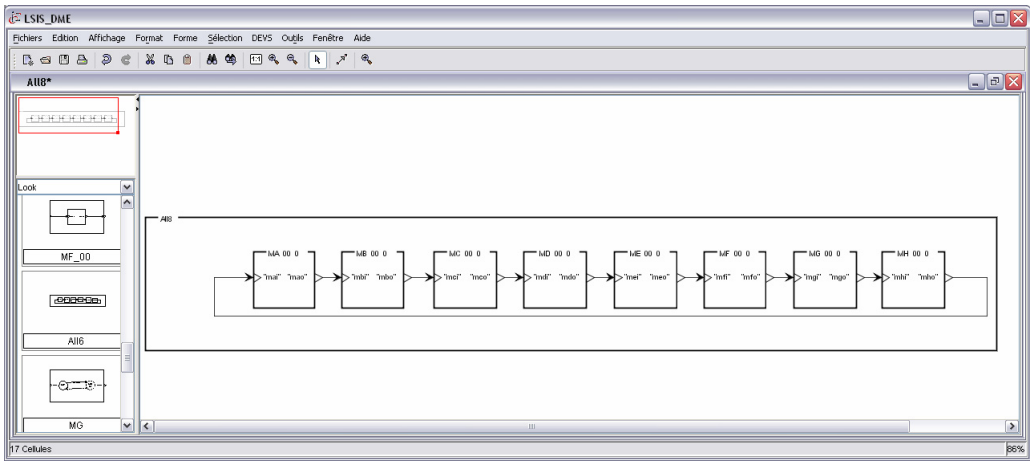


Figure 74 - Modèle couplé à 8 modèles

Nous avons exécuté des fédérations de modèles G-DEVS couplés, de 2, 4, 6 et 8 fédérés modèles G-DEVS distribués alternativement sur les deux ordinateurs décrits précédemment

(cf. Figure 75). L'exécution de ces fédérations a eut pour objectif de mesurer l'influence du Lookahead sur le temps d'exécution. Le code a été développé en Java.

Le RTI était exécuté sur un troisième ordinateur du réseau local (cf. Figure 75), dont les caractéristiques étaient identiques à celles des ordinateurs contenant les fédérés. Le logiciel retenu était la version « v2.3 r2 build 103 Certified for IEEE 1516 » de pRTI 1516 de la société [PITCH 02].

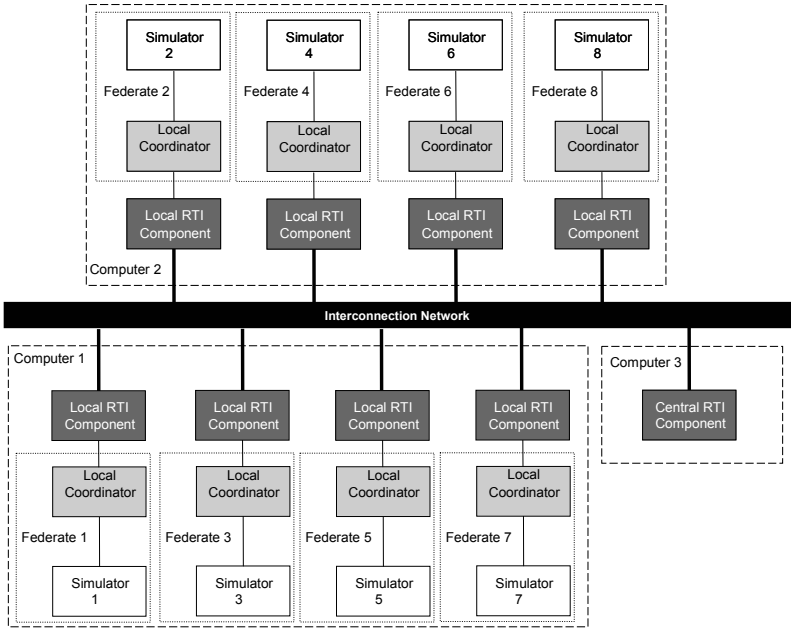


Figure 75 - Modèle couplé distribué à 8 modèles G-DEVS fédérés

Nous communiquons ci-dessous, quelques données clefs de ces simulations. Le tableau de la Figure 76 rapporte le volume d'informations échangées. Plus précisément, il fournit le nombre d'événements traités localement par un simulateur fédéré en rapport avec le nombre d'événements reçus de la fédération provenant d'un fédéré influenceur. Ce tableau fait apparaître que l'ordre de grandeur du nombre d'événements traités localement est comparable à celui du nombre d'événements reçus de la fédération. Ces résultats confirment donc l'équilibre des traitements locaux / distribués et valident ainsi la pertinence des modèles choisis pour effectuer des mesures de performances dans une simulation distribuée compatible HLA.

	2 fédérés	4 fédérés	6 fédérés	8 fédérés
ReceiveInteraction()	333	136	88	65
Nombre total de messages traités/ fédéré	169+333	137+138	88+89	65+68

Figure 76 - Nombre de messages traités par fédéré en fonction de la taille de la fédération

La Figure 77, issue de [ZACHAREWICZ 06a], présente les durées d'exécution pour des fédérations de tailles variables, en fonction de la valeur du Lookahead définie par les fédérés. Ces résultats sont calculés à partir de la moyenne d'une série de 10 réplications des mêmes fédérations de modèles couplés G-DEVS.

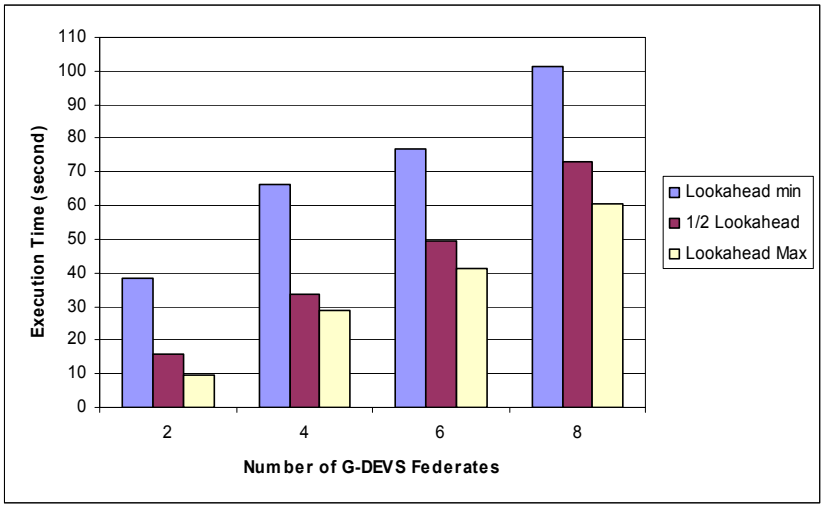


Figure 77 - Temps d'exécution en fonction du Lookahead

L'exécution de la fédération G-DEVS est accélérée en utilisant un Lookahead maximal (calculé à partir de la durée de vie des phases des modèles G-DEVS comme spécifié dans les § 3.4.1 et 3.4.2 du chapitre précédent). Cette affirmation est faite en comparaison des temps d'exécution des mêmes modèles fédérés avec un Lookahead réduit de moitié (représentant la première solution proposée dans § 2.3 avec une valeur de Lookahead unique) et avec un Lookahead négligeable (min) (représentant les solutions précédentes rappelées dans § 3.1).

4.3.2 Réduction du surcoût en temps de simulation

L'expérience déduit, également, que l'amélioration du rendement est proche d'une tendance linéaire en nombre de fédérés. Ainsi, les fédérés avec une valeur de Lookahead négligeable produisent toujours un important surcoût en temps d'exécution quant aux fédérés avec un Lookahead maximal. Ce surcoût augmente avec la taille de fédération, la Figure 78 présente le surcoût généré entre une exécution avec un Lookahead négligeable et un Lookahead maximal par la courbe OverHead 1 et le surcoût entre un Lookahead moyen et un Lookahead maximal par la courbe OverHead 2. Nous pouvons donc conclure, de la tendance de ces résultats, que l'utilisation d'un Lookahead négligeable ralentit significativement la simulation comme exposé théoriquement (jusqu'à une trentaine de secondes pour 8 fédérés).

Figure 78 - Surcoût en fonction de la valeur de Lookahead

Ces résultats sont basés sur la comparaison de fédérés utilisant des simulateurs locaux identiques. Pour cette raison les améliorations sont relatives à l'utilisation d'un même simulateur local, nous ne pouvons pas conclure à une amélioration absolue mais la tendance obtenue et les autres études relatives à l'utilisation du Lookahead abondent dans le même sens et permettent de corroborer l'étude réalisée dans ce mémoire.

5 Perspectives

Des travaux visant à l'amélioration de l'environnement LSIS_DME sont en cours. Un axe actuel de recherche et de développement autour de ce projet consiste à définir un format d'entrée des modèles sous forme textuelle, indépendant de l'éditeur. En effet, certains utilisateurs experts ont émis le souhait de pouvoir définir des modèles, dont les caractéristiques sont modélisables sous forme d'algorithmes de programmation, en particulier pour la définition des fonctions de transition, et la définition de certains modèles à états implicites.

Enfin, un deuxième axe concerne l'intégration de LSIS_DME dans un environnement de Workflow, en tant qu'outil de modélisation, de simulation et de supervision. Cet axe sera développé au cours du quatrième chapitre de ce mémoire.

6 Conclusions

Nous avons introduit dans ce chapitre plusieurs aspects de l'environnement LSIS_DME permettant la modélisation et la simulation DEVS et G-DEVS. Cet outil possède les atouts du formalisme G-DEVS en particulier, la clarté et la sûreté de ce formalisme, ainsi que l'efficacité de la simulation dirigée par les événements. De plus, il sépare, comme spécifié dans DEVS et G-DEVS, l'édition des modèles et leurs simulations, rendant ainsi la mécaniques de simulation indépendante du modèle lui même. En outre, le développement du programme en langage Java offre une portabilité de ce dernier sur les systèmes d'exploitation les plus répandus. Enfin la définition d'une « extension HLA » du simulateur permet de créer et de simuler des modèles G-DEVS distribués, intégrés dans des fédérés HLA. Cette extension donne également la capacité à l'environnement de communiquer avec d'autres programmes hétérogènes compatibles HLA.

Partant d'hypothèses établies par des calculs de complexités, l'efficacité de la mise à plat des modèles et du calcul du Lookahead relatif à l'état actuel du modèle a été mesurée et validée en termes de performance temporelle. Le gain ainsi obtenu atteint, en moyenne, un tiers dans la simulation séquentielle de modèles mis à plat pour un nombre constant d'événements. Le gain pour une simulation distribuée utilisant un Lookahead dépendant de l'état courant du modèle peut atteindre quarante pourcents dans les modèles distribués mis en œuvres.

Cet environnement a été utilisé dans le cadre de plusieurs projets de modélisation ainsi que pour des enseignements universitaires. LSIS_DME est référencé sur [WAINER 04] et il est disponible sur Internet[51].

[51] http://www.site.uottawa.ca/~oren/SCS_MSNet/coop-2005.htm

Chapitre 4. Application à un environnement Workflow G-DEVS / HLA

1 Introduction

L'objectif de ce chapitre est de proposer un environnement pour la modélisation et simulation des Workflow de production. Cet environnement est basé sur une représentation graphique et une modélisation en G-DEVS des procédures. De plus, il permet un interfaçage entre les différentes fonctionnalités du Workflow en se conformant à la norme HLA.

Après avoir sélectionné les concepts essentiels à l'élaboration d'un Workflow, nous présentons dans une première étape un langage de description créé pour définir les composants basiques d'une procédure Workflow (cf. WFMC[52]). Nous définissons, ensuite, une collection de modèles atomiques G-DEVS intervenant dans la modélisation d'une procédure Workflow. Nous présentons également l'algorithme permettant d'instancier les modèles atomiques et de générer leurs couplages pour définir un modèle couplé de la procédure Workflow. Ces modèles et leurs simulateurs sont intégrés à un environnement Workflow distribué, l'interfaçage entre les composants du Workflow étant assuré par le respect de la norme HLA. Enfin, nous illustrons l'utilisation de cet environnement par un exemple de Workflow de production de circuits électroniques.

2 Présentation du cadre de travail

2.1 Rappel du modèle Workflow de référence

La Figure 79, introduite par la WfMC [WFMC03 95], présente le moteur de Workflow qui assure l'interaction entre les différentes interfaces d'un Workflow. Nous situons, l'essentiel de l'étude développée dans ce chapitre, au niveau de l'outil de définition de procédures (*Build-Time*) et de l'interface 1 avec le moteur Workflow. Un cadre en transparence situe l'étude sur la Figure 79.

[52] WorkFlow Management Coalition

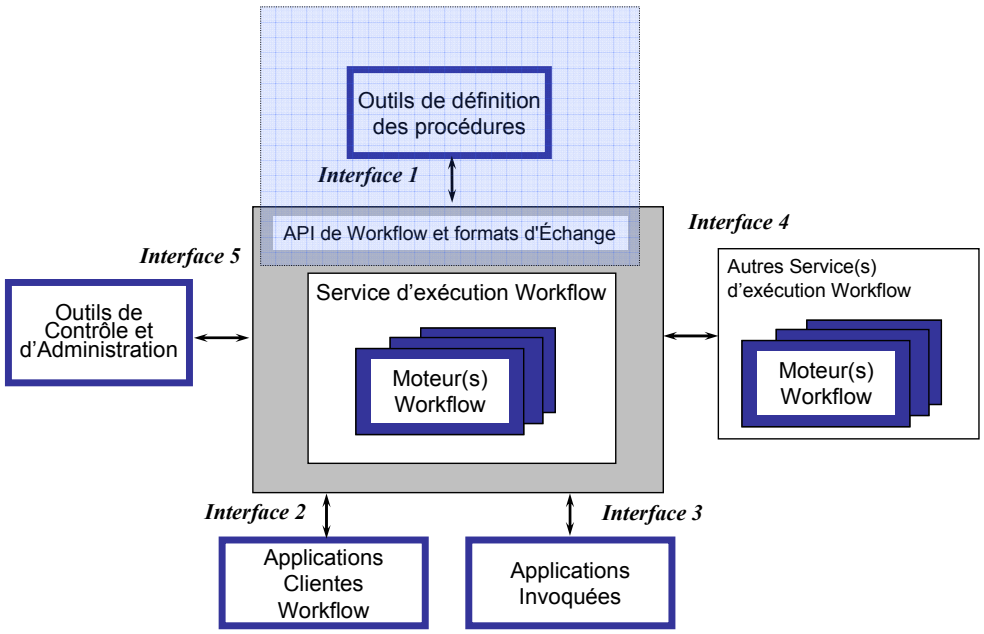


Figure 79 - Diagramme du Modèle de Référence Workflow

2.2 Détails de l'interface 1

Les outils proposés par le système de gestion de Workflow au niveau de l'interface 1 du modèle de référence sont consacrés à la définition des procédures. Plus en détails, ces outils permettent la définition, la modélisation, et l'analyse des procédures [WFMC25 05] (voir Figure 17 Chapitre 1).

A partir d'un modèle généré avec ces outils de modélisation, le moteur Workflow est capable de créer une version simulable du modèle. Le moteur Workflow peut alors exécuter une simulation de cette version et utiliser les résultats générés, pour interagir avec des applications de supervision et de monitoring au travers de l'interface 5. Dans ce cas, les résultats permettent, notamment, une comparaison, au travers de relevés, entre les résultats de simulation et les objectifs souhaités pour la procédure réelle. Le moteur peut également déclencher, en fonction de l'évolution des résultats de simulation, l'utilisation d'applications spécifiques au travers de l'interface 2 ou de l'interface 3, afin de traiter les informations par des applications externe ou par l'utilisateur.

En analysant les informations résultantes de la simulation du modèle de procédure (qui intègre également les données provenant des applications invoquées et celles communiquées par les acteurs), l'utilisateur du Workflow peut décider de modifier le modèle de la procédure en fonction d'un objectif modifié ou non atteint. Nous présentons dans la suite un processus visant à générer un modèle simulable à partir d'un modèle graphique de procédure Workflow.

3 Transformation d'une Structure Workflow en Structure G-DEVS

3.1 Processus de transformation Workflow G-DEVS

Nous définissons, dans cette section, un processus de transformation d'un modèle graphique de procédure Workflow vers une version simulable de ce modèle. Ce processus est illustré par la Figure 80.

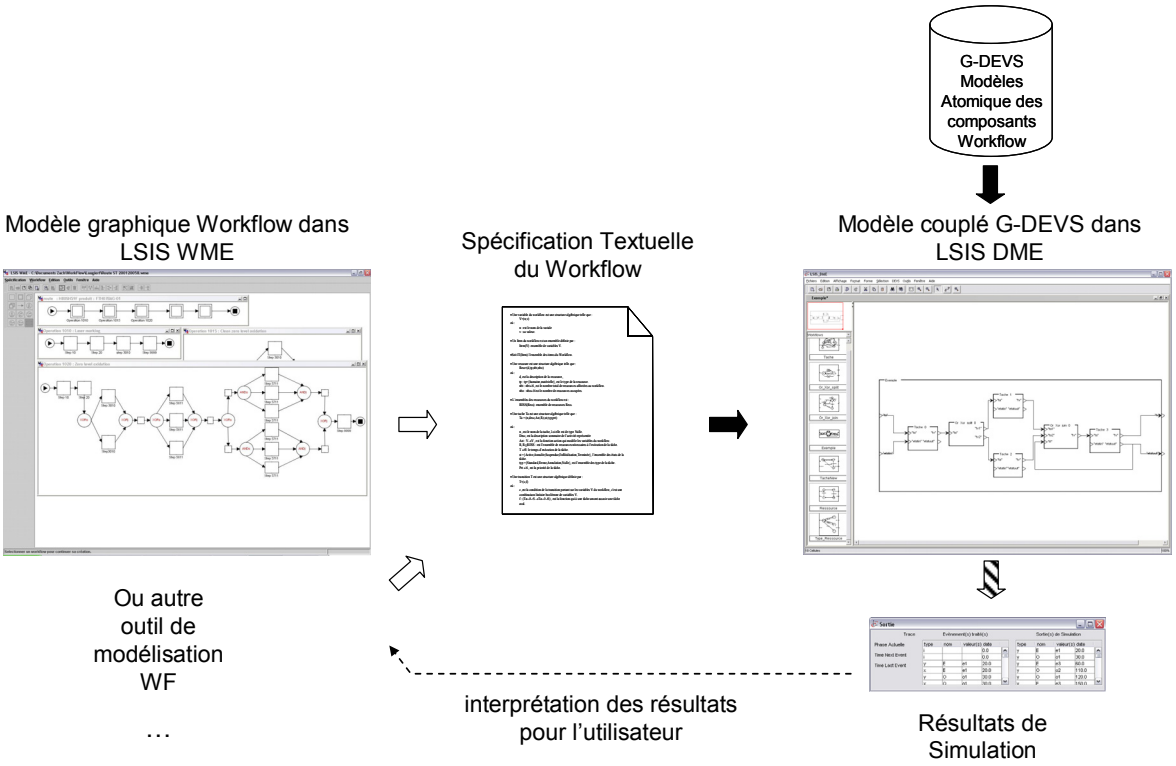


Figure 80 - Depuis un éditeur de Workflow vers un modèle G-DEVS

Dans le processus de transformation, les modèles sources sont issus d'un outil de modélisation de procédures Workflow. Nous avons choisi d'utiliser comme source de modèles graphiques, l'outil de modélisation de Workflow LSIS_WME[53] développé au sein du laboratoire LSIS. Le choix est porté sur cet outil car il est représentatif des outils existants, en effet, il repose sur les concepts énoncés par la WFMC et permet de représenter graphiquement le modèle d'une procédure Workflow.

Dans la première étape du processus (Flèches blanches sur la Figure 80), le modèle graphique est transformé dans un format textuel, indépendant de l'outil de modélisation. Cette modélisation est définie comme langage pivot.

Dans la deuxième étape (Flèches noires sur la Figure 80), le modèle en langage textuel de la procédure Workflow est interprété afin de créer un modèle simulable. Nous avons retenu, pour définir les modèles simulables, le formalisme G-DEVS, possédant une sémantique exécutable. De plus, afin de faciliter la génération de ces modèles, nous proposons de définir une

[53] LSIS Workflow Management Editor

base de modèles atomiques G-DEVS représentant les composants de base du Workflow (nous expliciterons par la suite chaque composant). Ces modèles atomiques seront instanciés à partir de paramètre fixés dans les modèles Workflow et couplés à partir des définitions issues du modèle textuel de procédure Workflow. La mise en œuvre a été réalisée par l'ajout d'une extension au moteur de simulation LSIS_DME, pour l'interprétation des modèles textuels.

Enfin, dans la troisième étape (Flèche hachurée sur la Figure 80), le modèle G-DEVS couplé est simulé. La trace de simulation est interprété pour fournir des résultats à l'utilisateur de l'outil de modélisation de Workflow, cette interprétation évite à l'utilisateur une connaissance du formalisme G-DEVS.

En détails, nos travaux se sont axés dans ce processus autour de la définition du langage textuel et de la transformation d'un modèle Workflow textuel en modèle couplé G-DEVS. Nous présentons dans les sections suivantes le langage textuel définissant un modèle Workflow et les algorithmes de transformation des modèles Workflow en modèles couplés G-DEVS.

3.2 Spécification textuelle d'un Workflow

3.2.1 Nécessité d'une spécification textuelle

Les environnements « Workflow », également appelés moteurs Workflow, proposent un outil permettant la modélisation et la simulation de modèles Workflow. Le formalisme de spécification des modèles est généralement ad hoc ou issu de méthodes classiques (par exemple les Réseaux de Petri) avec des spécificités liées à la définition d'un Workflow. Ce formalisme est, dans la majorité des applications, difficilement identifiable derrière l'outil graphique mis à disposition du modélisateur. Ces environnements ne sont donc pas toujours sûrs et transparents. En effet, les environnements dont les formalismes ne reposent pas sur une spécification formelle ne permettent pas d'effectuer des vérifications et des validations syntaxiques et sémantiques des modèles. De plus, il n'est pas évident d'importer des modèles depuis un autre environnement, et réciproquement, d'exporter un modèle pour le simuler avec un autre moteur Workflow basé sur un formalisme sous-jacent différent. Enfin, les données sources des modèles sont souvent ininterprétables lorsqu'elles sont sorties du contexte d'utilisation défini par l'environnement.

Pour apporter une réponse à ces problèmes, nous proposons d'introduire une représentation textuelle des modèles Workflow. Les modèles résultant seront donc lisibles indépendamment de l'outil utilisé pour représenter la procédure Workflow. Cette forme de modèle est donc portable. De plus, nous proposons de baser cette forme textuelle sur les concepts mathématiques de la théorie des ensembles pour donner la possibilité d'effectuer des vérifications de la validité de la structure des modèles à partir de propriétés issues de ces concepts.

3.2.2 Représentation d'une Procédure Workflow

Dans la suite de ce chapitre, nous proposons d'isoler les concepts nécessaires à la définition textuelle d'un Workflow de type production. Ces concepts sont sélectionnés parmi ceux

présentés par la WfMC [WFMC11 99] qui propose une définition pour chaque mot clé essentiel dans la terminologie Workflow. La syntaxe retenue pour la présentation des termes est la suivante : les termes seront tout d'abord définis puis, leurs contextes d'usage et leurs principaux synonymes seront énoncés. Enfin nous présentons la spécification textuelle que nous proposons pour chacun de ces concepts clefs.

3.2.2.1 Définition d'une Procédure Workflow

Nous avons présenté dans le premier chapitre la définition d'une **procédure d'un Workflow** (souvent appelé simplement Workflow). Il s'agit d'un ensemble coordonné d'activités ou d'opérations qui sont reliées, en série ou en parallèle, dans le but d'atteindre un objectif commun. Une procédure Workflow est une procédure dont le déroulement est contrôlé par un Workflow. Nous considérerons, dans ce chapitre, les flux de biens dans le cadre d'un Workflow de production dont la structure est, par définition, déterminée a priori.

Une procédure Workflow est en général composée de plusieurs activités qui s'enchaînent dans le but de modéliser un flux de travail. Il est donc nécessaire de définir des blocs de bases représentant les activités ou les opérations qui serviront à la construction d'un modèle de Workflow par couplage de ces blocs élémentaires à l'aide de blocs de routage.

Il existe plusieurs synonymes aux termes associés aux procédures Workflow, et il est important d'en connaître les principaux. En effet, lors de la prise en compte d'un modèle pour le représenter de façon textuelle, ce modèle peut provenir de différents formalismes ou techniques de description. Il est donc important de savoir relier l'ensemble des termes pris en compte par un même concept. Nous pouvons citer par exemple les synonymes de procédures Workflow : flux de tâches, flux de travail, processus ou procédures de travail etc. Ils seront pris en compte par le même concept.

3.2.2.2 Spécification textuelle

Nous donnons ici, pour les termes essentiels définis par la WFMC [WFMC03f 98], une représentation textuelle en utilisant les concepts et termes mathématiques issus de la théorie des ensembles. Un Workflow est en définitive composé d'un ensemble de composants intervenant dans le flux d'information ou de matière à gérer, nous détaillerons chaque composant par la suite.

Un Workflow (**Process-Procédure**) W est défini par une structure algébrique telle que :

W = (Name, RESS, A, Ct, ARC, IT, ST).

Où:

Name, le nom du Workflow,

RESS, l'ensemble des ressources du Workflow,

A, l'ensemble des tâches du Workflow,

Ct, l'ensemble des contrôleurs du Workflow,

ARC, l'ensemble des arcs du Workflow,

IT, l'ensemble des items du Workflow,

ST, l'ensemble des stocks du Workflow.

3.2.3 Définition des éléments basiques liés à une Procédure Workflow

Nous présentons ci-dessous chacun des éléments liés aux procédures Workflow introduites dans la section précédente. Ces termes sont issus des glossaires [WFMC03f 98] et [WFMC11 99]. Seuls sont explicités ceux que nous avons jugés suffisamment important pour participer à l'environnement de M&S de Workflow que nous proposons dans ce chapitre.

3.2.3.1 Bon de Travail (Work Item)

Définition

Un bon de travail est la représentation du travail à effectuer dans le cadre d'une instance d'activité par un acteur du Workflow.

Une Procédure est une succession d'Activités. Un Cas de Procédure donne lieu à une succession d'Instances d'Activités. Chaque Instance d'Activité est exécutée par un Acteur du Workflow qui trouve dans un Bon de Travail la représentation de ce qu'il a à faire. Les principaux synonymes sont une tâche, une unité de travail, un élément de travail ou un objet de travail.

Spécification textuelle

Une Variable d'Item du Workflow est une structure algébrique telle que :

V = (n, v)

Où :

n : est le nom de la variable

v : le valeur de la variable (cette valeur peut être une liste de valeurs).

Un Item (**Work Item-Bon de travail**) du Workflow est un ensemble défini par :

Item = {V}+ est un ensemble de variables V.

IT = {Item}* est l'ensemble des items du Workflow.

3.2.3.2 Ressource, Acteur (Workflow Participant, Invoked Application)

Définition

Un **Acteur** du Workflow est une ressource (machine, logiciel, être humain ou être humain utilisant un programme ayant une interface utilisateur) qui exécute une activité. Par extension, un acteur est toute ressource qui exécute partiellement ou totalement le travail dévolu à une instance d'activité. L'acteur du Workflow peut être utilisé au travers du rôle qui lui est associé. La terminologie distingue deux rôles : organisationnel ou procédural. Nous nous concentrerons sur le rôle procédural et, par soucis de simplification, nous confondrons en conséquence le rôle procédural et l'acteur. Ces informations seront associées à ce que nous définissons comme les attributs type et sous-type d'un acteur.

Un Acteur du Workflow peut travailler sur un ou plusieurs bons de Travail en même temps.

Nous nous sommes efforcés de respecter la terminologie en définissant pour chaque acteur du Workflow les états suivants :

Actif : Acteur du Workflow en fonction, en cours d'exécution d'au moins un bon de travail

Inactif : Acteur du Workflow en fonction, mais qui n'a pas de bon de travail à exécuter.

Dans l'environnement, les acteurs sont également modélisés afin de définir des Workflow de test. Ils pourront également être des entrées du modèle dans le cas d'une utilisation avec un « humain dans la boucle ».

Un acteur dans notre environnement de modélisation de Workflow peut être :

- une machine ; le Workflow doit être en mesure de s'interfacer avec des informations provenant de machines,
- un être humain ; nous modélisons dans ce cas le comportement humain pour les modèles de test de Workflow
- un programme ; cela peut être un programme utilisé dans le Workflow ou un modèle d'un acteur humain ou machine,
- un être humain utilisant un programme, c'est-à-dire ce qui est décrit par l'interface 2.

Plusieurs Acteurs du Workflow peuvent exister pour une activité donnée. Par exemple, deux numériseurs peuvent travailler en parallèle pour exécuter l'activité de numérisation des bordereaux de commande. Nous avons introduit la notion de performance souvent négligée dans la modélisation des acteurs du Workflow. En effet les acteurs ont dans la majorité des cas une performance binaire (en service/ hors service). Cependant, cette performance influence l'accomplissement d'une tâche, la prise en compte d'une notion de performance plus complexe (prenant en considération, par exemple, le comportement humain ou le comportement défaillant d'une machine) peut nous aider à définir des modèles plus précis et réalistes.

Les principaux synonymes sont : un agent, un intervenant, un participant, un traitant ou un utilisateur.

Spécification textuelle

Une ressource (Workflow Participant-Acteur, Invoked Application et Outil Workflow) est une structure algébrique telle que :

Ress = (n, tp, stp, Perf)

Où:

n, est le nom de la ressource.

tp, tp $\in$ {humaine, matérielle}, est le type de la ressource,

stp, stp $\in$ {opérateur, contrôleur, machine à découper, machiner à assembler…} est le sous-type de la ressource,

Perf : {{stp} $\times$ {t / t$\in \Re$}} $\rightarrow$ {normale, x % de sa capacité…} est la fonction qui définie les performances de la ressource,

L'ensemble des ressources du Workflow est :

RESS = {(Ress, nb)*} ensemble de ressources Ress.

nb : nb $\in \aleph$, est le nombre de ressources par description affectées au Workflow.

3.2.3.3 Activité (tâche) WorkFlow (WorkFlow Process Activity)

Définition

Une **activité** (ou une **tâche**) est une étape logique d'une procédure au cours de laquelle une action élémentaire est exécutée au niveau du Workflow considéré. Une **activité Workflow** est une activité qui fait partie d'une procédure Workflow. Une activité peut également être un processus Workflow dans le cas de Workflow composables modulaires. La réalisation de tâches contribue à l'accomplissement d'une procédure Workflow. Notons qu'il existe deux tâches particulières : la tâche de début et la tâche de fin.

La tâche peut être par exemple la reconnaissance optique des caractères d'un document numérisé, Étape de validation d'une chaîne de saisie de données, etc.

Une procédure Workflow est en général composée d'une ou plusieurs activités Workflow définissant, une fois connectées, les itinéraires possibles de la procédure. De plus dans un but de représentation modulaire et hiérarchique du Workflow, nous considérons qu'une tâche peut être une tâche atomique ou un sous-Workflow.

Spécification textuelle

Une tâche atomique (**Process Activity-Activité**) Ta est une structure algébrique telle que :

Ta = (n, desc, Act, R, Texec, Et, Prt)

Où :

n, est le nom de la tâche.

Desc, est la description sommaire de l'activité représentée

Act : Item $\rightarrow$ Item, est la fonction action qui modifie les Items du Workflow.

R, R $\subseteq$ RESS : est l'ensemble de ressources nécessaires à l'exécution de la tâche.

Texec : {PerfRessource $\times$ {Item}} $\rightarrow \Re$: fonction qui définit le temps d'exécution de la tâche pour un item donné.

Et = {Libre, Demande Allocation, Attente réponse Ressource, Ressource non Disponible, En cours d'exécution, Allocation ressource} est l'ensemble des phases de la tâche.

Prt : {Item} $\rightarrow \aleph$, est la fonction priorité de la tâche.

Sta : {ST} $\rightarrow$ St fonction qui associe un stock à la tâche

Une tâche composite (**Sub Workflow, Sub Process- Sous-Procédure**) Tc est un Workflow.

L'ensemble A des tâches du Workflow est :

A = {Ta, Tc}* ensemble de tâches qui définissent un Workflow.

3.2.3.4 Itinéraire (Route)

Définition

L'**itinéraire** d'un cas de procédure est la suite des activités qui ont été traversées lors de l'exécution de ce cas précis. Une procédure permet en général plusieurs itinéraires possibles.

Nous avons choisi par rapport à la définition WfMC, de décomposer la route par un ensemble d'arcs élémentaires entre blocs de tâche ou de contrôle. Une route sera donc définie par un sous ensemble des arcs du Workflow.

Spécification textuelle

Un arc (**Route-Itinéraire**) Arc est une structure algébrique définie par :

Arc = (n, f)

Où :

n : $n \in \aleph$, référence de l'arc

f : {A, Ct} $\rightarrow$ {A, Ct}, est la fonction qui associe, à une tâche ou un contrôleur amont, une tâche ou un contrôleur aval.

L'ensemble ARC des arcs du Workflow est :

ARC = {Arc}*, définit l'ensemble des arcs.

3.2.3.5 Condition de transition, routes (Transition Condition)

Définition

Une **condition de transition** est le critère de progression, ou de changement d'état d'une activité (étape de travail) à l'activité (étape) suivante lors d'un cas d'exécution donné, qu'il s'agisse d'une procédure manuelle ou informatisée. Il est important de préciser que nous modélisons, dans notre étude, des Workflow de production dont la structure est plutôt statique. Le routage des items, dans ces cas, est effectué en définissant des conditions sur les items, contrairement aux Workflow administratifs où le routage est souvent défini par la ressource.

Nous pouvons illustrer une condition de transition par l'exemple suivant : une procédure de déclaration de sinistre du client X est en transition entre l'activité expertise et l'activité remboursement parce que la condition de transition est respectée.

Les conditions de transition peuvent être décrites lors de la modélisation des procédures et/ou être calculées par le système de gestion de Workflow pendant l'exécution des procédures.

Les principaux synonymes sont contrôleurs de routage, conditions de branchement ou encore règles de routage / de circulation.

Spécification textuelle

Cette spécification est décomposée dans chaque type de contrôleurs.

L'ensemble Ct des contrôleurs du Workflow est :

Ct = {So, Sa, Sx, Jo, Ja, Jx, It}*, définit l'ensemble des contrôleurs.

3.2.3.6 Séquence (Sequential Routing)

Définition

Le routage d'une procédure est défini en **séquence** si ses activités sont exécutées les unes à la suite des autres, et s'il existe un seul itinéraire possible.

Une illustration d'une séquence est : un bon de commande traité en une séquence de trois activités consécutives.

Les principaux synonymes sont acheminement séquentiel, enchaînement, exécution ou routage séquentiel.

Spécification textuelle

Un enchaînement séquentiel est simplement représenté par un arc entre deux blocs tâches.

Un arc (**Route-Itinéraire**) Arc est une structure algébrique définie par :

Arc = (n, f)

Où :

n : $n \in \aleph$, référence de l'arc

f : {A, Ct} $\rightarrow$ {A, Ct}, est la fonction qui associe, à une tâche ou un contrôleur amont, une tâche ou un contrôleur aval.

3.2.3.7 Aiguillage (OR-Split & XOR-Split)

Définition

On parle d'**aiguillage** lorsqu'un itinéraire s'ouvre sur plusieurs itinéraires possibles et que le cas d'exécution suit l'un OU l'autre de ces itinéraires, selon les conditions de transition. Le (ou les) items produit(s) en sortie sont générés en fonction de condition reposant sur le contenu de l'item. Notons que ce choix diffère des Workflow administratifs où le routage est majoritairement défini par les acteurs.

Les principaux synonymes sont acheminement, branchement, enchaînement, routage conditionnel, ou encore, bifurcation.

Spécification textuelle

Un contrôleur **Or-Split** So est une structure algébrique définie par :

So = (i, o, fos)

Où :

i : $i \in$ IT, est l'entrée du contrôleur Or-Split.

o : $o \subseteq$ IT, est l'ensemble des sorties du contrôleur Or-Split.

fos : $i \rightarrow o$, est la fonction qui transmet l'item de l'entrée i du contrôleur Or-Split, à toute sortie o dont la condition d'enchaînement est vérifiée.

$fos(Item_i) = Item_{iverif} = Item_{jverif}$…

Un contrôleur **Xor-Split** Sx est une structure algébrique définie par :

Sx = (i, o, fxs)

Où :

i : i ∈ IT, est l'entrée du contrôleur Xor-Split.

o : o ⊆ IT, est l'ensemble des sorties du contrôleur Xor-Split.

fxs : i → o, est la fonction qui transmet l'item de l'entrée i du contrôleur Xor-Split, à la sortie o dont la condition d'enchaînement est vérifiée.

$fxs(Item_i) = Item_{iverif}$

3.2.3.8 Jonction (OR-Join & XOR-Join)

Définition

On appelle **jonction,** la convergence de deux ou plus itinéraires vers une même activité. Il ne s'agit pas de la synchronisation de plusieurs itinéraires, mais plutôt de la jonction de plusieurs itinéraires alternatifs. Dans le cas du XOR un seul item est transmis en fonction de condition sur les items entrants. Le principal synonyme est convergence.

Spécification textuelle

Un contrôleur **Or-Join** Jo est une structure algébrique définie par :

Jo = (i, o, foj)

Où :

i : i ⊆ IT, est l'ensemble des entrées du contrôleur Or-Join.

o : o ∈ IT, est la sortie du contrôleur Or-Join.

foj : i → o, est la fonction qui pour tout élément i en entrée associe la sortie o si un Item est reçu.

$foj(Item_{i1}…Item_{in}) = Item_o$

Un contrôleur **Xor-Join** Jx est une structure algébrique définie par :

Jx = (i, o, fxj)

Où :

i : i ⊆ IT, est l'ensemble des entrées du contrôleur Xor-Join.

o : o ∈ IT, est la sortie du contrôleur Xor-Join.

fxj : i → o : est la fonction qui, pour tout élément i en entrée, associe la sortie o sur laquelle l'Item est transmis, si plusieurs entrées reçoivent un Item simultanément, un seul est recevable et transmissible, les autres sont ignorés.

$fxj(Item_i) = Item_o$.

3.2.3.9 Branchement multiple (AND-Split)

Définition

Il y a **branchement multiple** lorsqu'un itinéraire unique se sépare en deux ou plusieurs itinéraires différents dans le but de réaliser deux ou plusieurs activités en parallèle. Il se peut que pour un cas de procédure donné, il y ait plusieurs instances d'activité en cours, par exemple si l'itinéraire suivi par le cas de procédure comporte un branchement multiple. A la suite de ce branchement, au moins deux instances d'activités peuvent être exécutables.

Dans ce cas, plusieurs corbeilles peuvent contenir chacune un bon de travail décrivant ce qu'il faut faire dans chacune des deux (ou plus) instances d'activité en rapport avec ce cas de procédure

Les principaux synonymes sont éclatement, routage multiple ou routage en parallèle.

Spécification textuelle

Un contrôleur **And-Split** Sa est une structure algébrique définie par :

Sa = (i, o, fas)

Où :

i : $i \in IT$, est l'entrée du contrôleur And-Split.

o : $o \subseteq IT$, est l'ensemble des sorties du contrôleur And-Split.

fas : $i \rightarrow o$, est la fonction qui transmet l'item de l'entrée i du contrôleur And-Split à toutes les sorties o du contrôleur.

$fas(Item_i) = Item_{o1} = Item_{o2} = \ldots Item_{on.}$

3.2.3.10 Rendez-vous (AND-Join)

Définition

Il y a **rendez-vous** lorsque deux ou plusieurs activités parallèles convergent vers un itinéraire unique et que l'on assure la synchronisation des itinéraires, c'est-à-dire qu'on ne passera à l'activité suivante que lorsque toutes les activités parallèles seront achevées.

Les principaux synonymes sont Jonction synchronisée, Synchronisation ou Recollement.

Spécification textuelle

Un contrôleur **And-Join** Ja est une structure algébrique définie par :

Ja = (i, o, faj)

Où :

i : $i \subseteq IT$, est l'ensemble des entrées du contrôleur And-Join.

o : $o \in IT$, est la sortie du contrôleur And-Join.

faj : $i \rightarrow o$, est la fonction qui associe à n éléments i en entrée, quand les n items différents sont reçus, une sortie o où l'item est transféré.

$faj(Item_{i1}, Item_{i2}, \ldots Item_{in}) = Item_o$

3.2.3.11 Itération (Iteration)

Définition

Une **itération** est un cycle d'activité(s) qui implique la répétition d'une même ou de plusieurs mêmes instances activités jusqu'à ce qu'une condition soit remplie.

Les principaux synonymes sont boucle ou répétition.

Spécification textuelle

Un contrôleur **Iteration** It est une structure algébrique définie par :

It = (i, o, fit)

Où :

i : $i \in$ IT, est l'entrée du contrôleur Iteration.

o : $o \in$ IT, est la sortie du contrôleur Iteration.

fit : $i \rightarrow o$, est la fonction qui transmet l'item de l'entrée i du contrôleur Iteration (sortie de la tâche amont), à la sortie o connecté à l'entrée de la tâche amont.

$fit(Item_i) = Item_i$

3.2.3.12 Stock, Corbeille (Worklist)

Définition

Une **Stock** (**Corbeille)** contient une liste de Bons de Travail à exécuter par un Acteur du WorkFlow. Généralement dans la modélisation Workflow, les stocks sont associés aux acteurs du Workflow sous forme de liste de tâches. Nous avons choisi de positionner en amont des tâches ces composants Workflow de production car ces derniers sont plus orientés items (produits) qu'acteurs (ressources humaines ou matérielles).

Les principaux synonymes sont Liste de tâches ou Liste bons de travail.

Spécification textuelle

Un stock St (**~WorkList-Corbeille**) est une structure algébrique telle que :

St = (n, Cap, Pr)

Où :

n, est le nom du stock.

Cap : $\{\aleph \times t\} \rightarrow \Re$, est la fonction qui définit la capacité du stock.

Pr : $\{\aleph\} \rightarrow$ {FIFO, LIFO, Priorité particulière, Round Robin} est la fonction qui définit les règles de service du stock.

L'ensemble ST des stocks du Workflow est :

ST = {St}* ensemble de stocks St.

3.3 Définition de Modèles Workflow simulables dans le formalisme G-DEVS

3.3.1 Choix du formalisme des modèles simulables des Workflow

Le formalisme classiquement associé aux Workflow est les Réseaux de Petri. Cette technique est devenue une référence dans le domaine de la modélisation des Workflow notamment au travers des travaux de l'auteur [VAN DER AALST 98b]. Nous avons proposé une version distribuée de ce type d'environnement de modélisation et simulation de Workflow basé sur le formalisme des Réseaux de Petri et la norme HLA dans [ZACHAREWICZ 02]. Cependant, nous remettons ci-dessous en question la légitimité de l'utilisation de ce formalisme.

3.3.1.1 Trois raisons d'utiliser les réseaux de Petri

Un modèle de réseau de Petri est représenté par un cinq-tuple (P, T, I, O, M) [PETRI 62]. P est un ensemble fini de places (couramment symbolisé par des cercles) représentant les conditions. T est un ensemble fini de transitions (couramment symbolisé par une barre), qui représente les événements. I et O représentent un ensemble fini de fonctions de transitions faisant correspondre les transitions à l'ensemble des places. Enfin, M est l'ensemble initial de marquage des places. Les places peuvent contenir zéro ou plusieurs jetons à un instant donné.

[VAN DER AALST 98b] énonce les raisons de modéliser les Workflow avec les réseaux de Petri et évoque également la possibilité de s'en servir de modèle de référence pour la modélisation conceptuelle des Workflow.

Raison 1 : Une sémantique formelle.

Cet outil allie la facilité de lecture graphique et la sémantique formelle qui permet de réaliser toutes les configurations de réseaux primitifs des Workflow.

Raison 2 : Un modèle basé sur les états et non sur les événements.

Une description basée sur les états permet de réaliser une distinction claire entre la disponibilité d'exécution d'une tâche et sa réelle exécution. Ceci est important lorsque l'on sait que les Workflow sont soumis à des déclencheurs externes.

Raison 3 : Une abondance de techniques d'analyse.

Les réseaux de Petri sont associés à un nombre important de techniques d'analyse qui permettent l'analyse des Workflow.

3.3.1.2 Trois raisons de préférer le formalisme DEVS

Le formalisme DEVS s'impose comme un formalisme pouvant inclure le formalisme des réseaux de Petri. De plus, ce dernier possède une puissance d'expression limitée. Pour parer à cela, un nombre important d'extensions a été nécessaire [ZEIGLER 02].

Raison 1 : Dans les réseaux de Petri basiques, la notion de temps n'est pas considérée.

Le temps n'est pas une variable des réseaux de Petri classiques.

Raison 2 : Les modèles des réseaux de Petri basiques ne sont pas modulaires.

Contrairement aux modèles DEVS le formalisme de base des réseaux de Petri ne permet pas une composition hiérarchique des modèles pour la définition de modèles composites.

Raison 3 : Les réseaux de Petri basiques ne sont pas adaptés à l'implémentation.

En effet, il n'est décrit aucune structure de simulation des réseaux de Petri contrairement au formalisme DEVS.

3.3.1.3 Principe de composition hiérarchique

Une des raisons du choix du formalisme DEVS, par rapport aux autres formalismes en général, réside dans la possibilité de définir des modèles hiérarchiques et modulaires. En effet, les modèles DEVS peuvent être définis par composition d'éléments de base ou d'éléments provenant du stock en bibliothèque. Cette représentation s'adapte donc à toutes les notions présentées par la WFMC, notamment concernant la nécessité de pouvoir créer des modèles Workflow hiérarchiques.

Plus en détails, nous proposons de créer un ensemble de modèles de bases représentant les différents composants, introduits dans la section précédente, d'un modèle Workflow. Nous choisissons le formalisme de modélisation G-DEVS afin de représenter les items par des événements « complexes » en utilisant les coefficients d'un polynôme. Chaque item contient une ou plusieurs valeurs d'état de traitement, un identifiant, une information de routage, etc. Ces modèles G-DEVS sont dits génériques et paramétrables car ils seront instanciés en fonction du cas d'un Workflow à traiter.

3.3.2 Définition d'une Bibliothèque de modèles G-DEVS pour la définition d'un Workflow

Nous proposons de définir tous les éléments basiques composant un Workflow par un modèle atomique G-DEVS équivalent. Ces modèles, stockés dans une base de modèles, devront être instanciés en fonction des valeurs des différents paramètres provenant de la structure Workflow. Nous présentons Figure 81 la bibliothèque de modèles G-DEVS pouvant être utilisés.

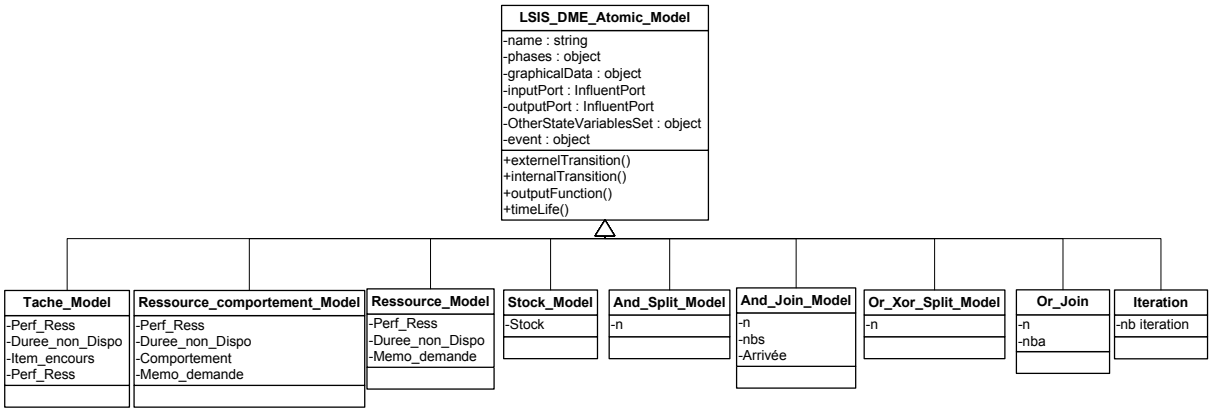


Figure 81 - Bibliothèque de modèles Workflow

Chaque modèle hérite des attributs et des fonctions de bases de la classe modèle atomique G-DEVS présentée précédemment, cependant certaines variables d'état et paramètres spécifiques sont définies pour chaque classe de sous-modèle afin et de décrire le comportement et de répondre aux paramétrages de chaque élément de composant un Workflow.

3.3.2.1 Modèles atomiques paramétrables

Les modèles G-DEVS des composants WF sont dits génériques et paramétrables, c'est-à-dire qu'ils comportent des caractéristiques communes. Certains paramètres restent pour autant à compléter à la charge du modélisateur final (celui qui représente un cas d'un Workflow).

Cette solution présente l'avantage pour l'utilisateur néophyte de lui permettre de modéliser un Workflow de façon rapide à partir de modèles « patrons ». Un utilisateur plus expérimenté pourra utiliser et paramétrer des modèles existants, mais également créer de nouveaux modèles spécifiques à ses besoins.

Nous allons présenter ci après les éléments de la bibliothèque de base permettant d'élaborer un modèle Workflow par composition de ces éléments. Le détail de chaque modèle est donné dans un premier temps, les règles de couplage sont énoncées par la suite.

Modèle atomique G-DEVS du composant Stock

Le Modèle atomique G-DEVS du composant Stock est composé de trois phases. Il a pour fonction d'empiler et de dépiler les items de travail qui arrivent sur une tâche.

Il a été fait ici le choix de gérer la notion de stock en amont des tâches et non pas en amont des ressources comme cela peut exister dans les environnements de gestions classiques de Workflow, par exemple [STAFFWARE 01]. Ce choix est motivé par la possibilité de gérer plus directement, dans ce type de représentation, les tâches et ainsi d'identifier plus facilement les goulots d'étranglement en amont des tâches sur le flux de travail. Cette représentation est plus orientée flux de produits ou d'informations que ressources.

Concrètement le Stock gère une liste contenant des objets (Item) en transit ne pouvant pas être pris en charge directement par la tâche en aval. Dans ce cas, l'objet est empilé et, lorsque la tâche en aval informe de sa nouvelle disponibilité, un objet est dépilé. A priori, la gestion de la pile est de type LIFO, mais il est possible de gérer le dépilement de façon différente en fonction des nécessités de la procédure.

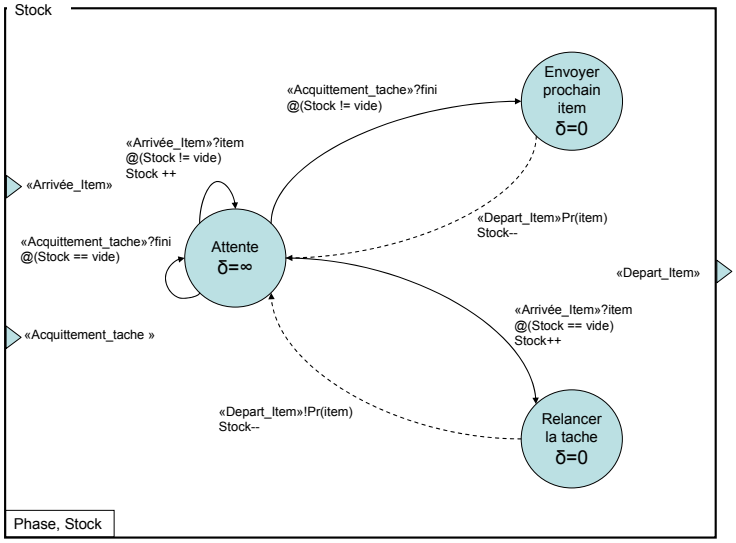


Figure 82 - Modèle atomique G-DEVS composant Stock

Un modèle de stock défini sera donc intercalé en amont de chaque tâche dans le but de gérer les items arrivant pendant le traitement d'un item précédent.

Modèle atomique G-DEVS du composant Tâche basique

Le modèle de tâche est initialement dans un état d'attente d'item à traiter. Suite à la réception d'un item, la tâche doit effectuer une demande d'allocation auprès d'une ressource. Si la ressource est disponible, la tâche peut effectuer le travail sur l'item puis le libérer. Si la ressource n'est pas disponible, l'item est mis en attente, et une nouvelle demande d'allocation sera effectuée auprès de la ressource après la durée d'occupation communiquée par cette dernière.

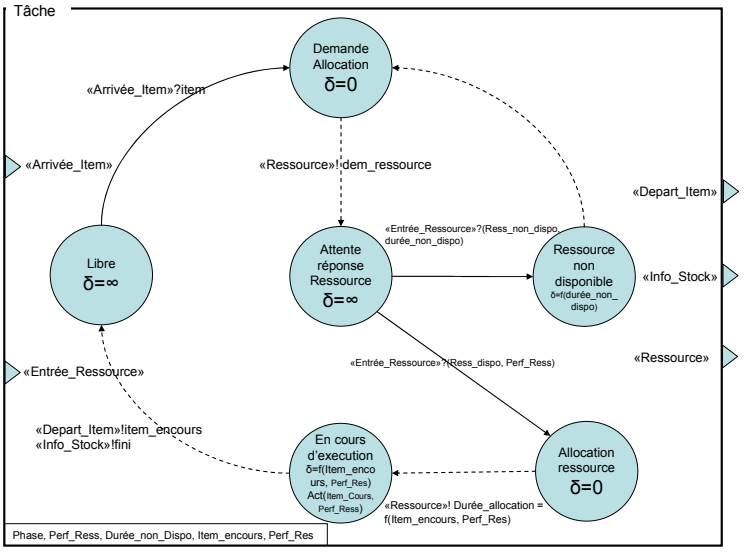


Figure 83 - Modèle atomique G-DEVS composant Tâche basique

Modèle atomique G-DEVS du composant Ressource

Le modèle de ressource est initialement en attente de demande d'allocation. Lorsqu'une demande est reçue dans cet état, la ressource répond qu'elle est disponible. Suite à cela, la

tâche qui souhaite l'utiliser lui communique la durée d'allocation. La ressource est alors occupée pour une durée donnée, et ne peut donc plus, dans cet état, être allouée à une autre tâche. En conséquence, les autres tâches effectuant une demande d'allocation recevront une réponse négative à leur requête. Lorsque la durée d'allocation est écoulée, l'état du modèle redevient libre et donc disponible à l'allocation par une tâche.

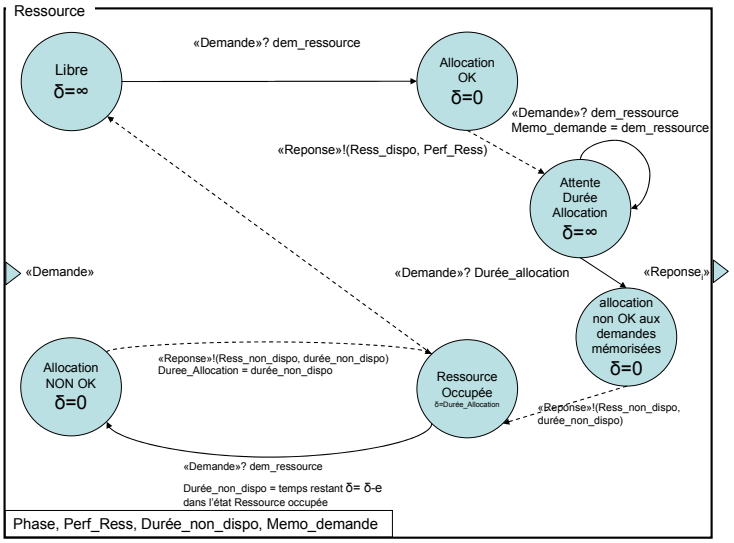


Figure 84 - Modèle atomique G-DEVS composant Ressource

Modèle atomique G-DEVS du composant Ressource avec comportement modélisé

Le modèle atomique de ressource avec comportement est similaire à celui présenté ci-dessus à l'exception de l'influence possible sur ce modèle d'un autre modèle de comportement. Nous distinguons deux types de modèles de comportement, les modèles de ressources matérielles et les modèles de ressources humaines.

Le modèle de comportements matériel est défini à partir des documents techniques de la machine utilisée, ces documents fournissent en particulier les taux de défaillance, la fiabilité et la disponibilité de ces ressources.

Le modèle de comportement humain est établi en fonction de facteurs psychologiques et physiologiques. Plus concrètement, des facteurs tels que les émotions, le stress, la fatigue pourront être pris en compte. Ces modèles s'appuient sur les travaux de [SECK 05]. Ces modèles pourront fournir des informations intéressantes pour maîtriser la sûreté de fonctionnement du Workflow modélisé.

Ces modèles permettent de rendre plus réaliste le modèle global du processus Workflow.

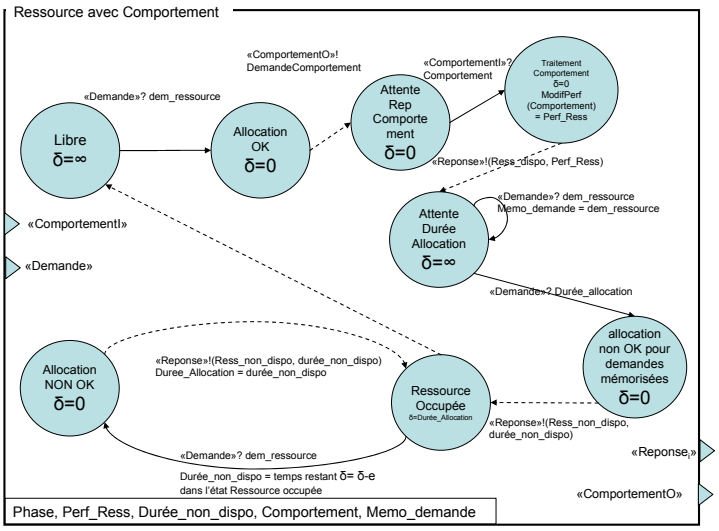


Figure 85 - Modèle atomique G-DEVS composant Ressource avec comportement

Modèle G-DEVS des composants de contrôle

Nous présentons chacun des contrôleurs utilisables pour modéliser un Workflow dans notre environnement.

Modèle G-DEVS du composant AND-SPLIT standard

Ce modèle décrit le contrôleur de disjonction And Split. Ce modèle reçoit des items sur son unique port d'entrée. Chaque item reçu est copié sur chacun des ports de sortie du modèle et diffusé par des événements de sortie.

Dans ce contrôleur, les items reçus en entrée sont copiés vers les sorties avec un délai δ négligeable ou nul et sans condition.

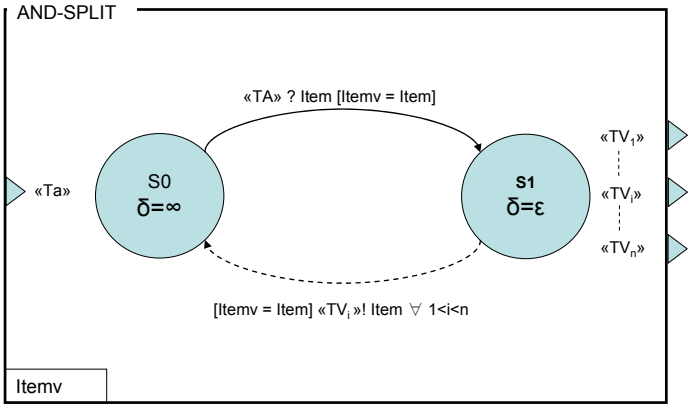


Figure 86 - Modèle G-DEVS du AND-SPLIT

Modèle G-DEVS du composant AND-JOIN

Ce modèle décrit le contrôleur de jonction And Join. Ce modèle reçoit des items sur ses ports d'entrées multiples. Les items reçus sont stockés dans une liste (variable d'état Tab_Item) jusqu'à ce que le contrôleur ait reçu un item sur chaque port (compteur nba = nbs) ; un item est alors généré sur le port de sortie unique.

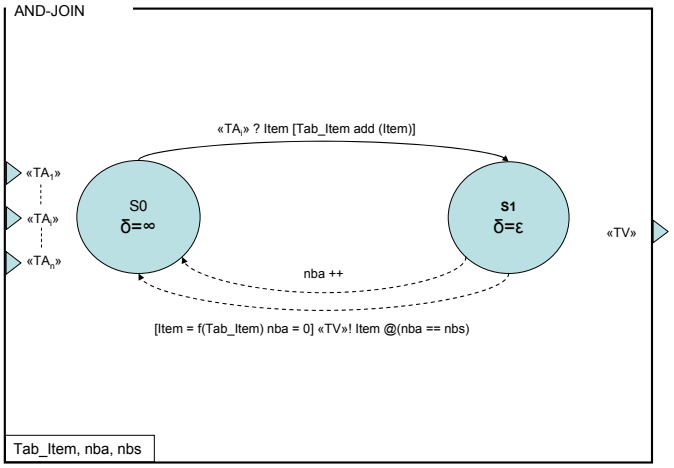


Figure 87 - Modèle G-DEVS du AND-JOIN

Modèle G-DEVS du composant OR/XOR SPLIT

Ce modèle décrit le contrôleur de disjonction Or/Xor Split. Ce modèle reçoit des items sur son unique port d'entrée. En fonction de conditions posées (C_i) pour chaque port de sortie (TV_i) sur le contenu de l'item reçu, le contrôleur va diriger l'item vers un ou plusieurs ports de sortie (TV_1, …, TV_i, …, TV_n).

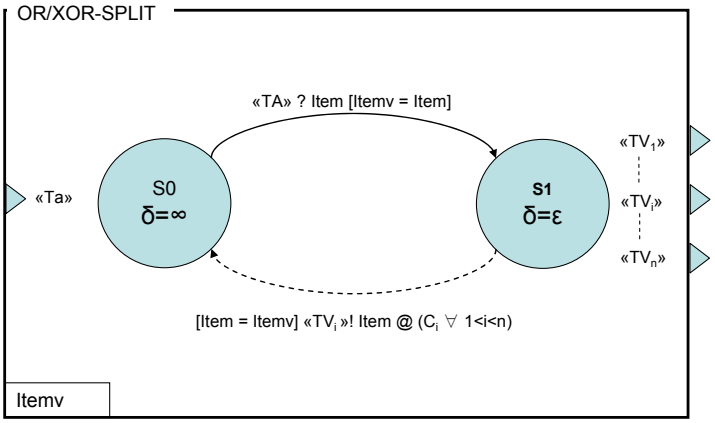


Figure 88 - Modèle G-DEVS du OR/XOR SPLIT

Modèle G-DEVS du composant Itération

Le modèle itération possède un compteur du nombre de passage d'un événement Item. Lorsque l'item est passé par ce modèle un nombre de fois souhaité (*nb iteration*), il est libéré vers la tâche suivante. Ce modèle trivial n'est pas représenté ici.

Modèle G-DEVS du OR/XOR-JOIN

Ce modèle décrit le contrôleur de jonction OR/XOR Join. Ce modèle reçoit des items sur ses ports d'entrées multiples. L'item de sortie est issu d'un calcul (f(Tab_Item)) sur les items reçus. Il est possible d'attendre l'arrivée d'un nombre donné (*nbs*) d'Item avant d'émettre un item de sortie. Notons que les items reçus simultanément peuvent être fusionnés dans un seul item alors généré sur le port de sortie unique. Eventuellement des règles visant à établir un XOR sur les items reçus peuvent être instaurées à partir des conditions C_1 à C_n.

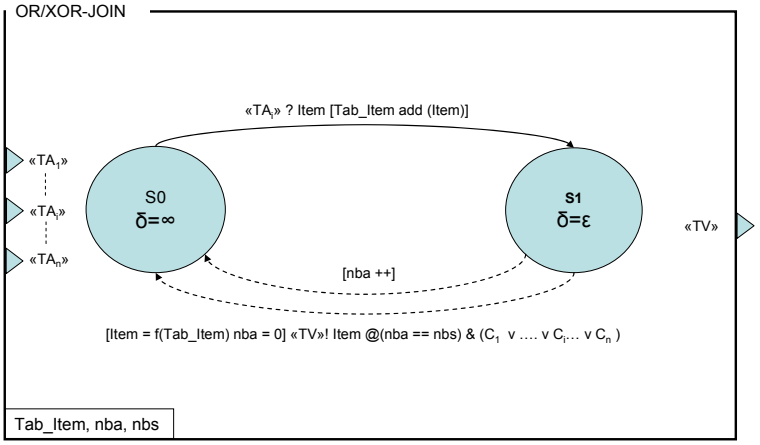


Figure 89 - Modèle G-DEVS du OR-JOIN avec fusion

3.3.2.2 Modèle couplé G-DEVS d'un Workflow

A partir des éléments de la bibliothèque nous pouvons représenter le modèle d'un processus Workflow complet en les couplant. Il faudra pour cela veiller à respecter certaines règles dans le couplage des composants, qui sont énoncées ci-dessous.

Règles de couplage

Une grammaire doit définir les règles de couplage entre les différents composants. Une BNF[54] est actuellement en développement au sein du laboratoire LSIS. Dans cette thèse, nous nous limitons à une présentation informelle des règles de couplages entre les composants.

Règle 1 : Tout modèle de Workflow contient un modèle de début et un modèle de fin.

Règle 2 : Tout modèle de tâche est connecté en amont à un connecteur.

Règle 3 : Tout modèle de tâche est connecté en aval à un connecteur.

Règle 4 : Tout modèle de tâche est connecté à un modèle de stock.

Règle 5 : Tout modèle de tâche est connecté à (au moins) un modèle de ressources.

Règle 6 : Tout modèle de connecteur est connecté en amont et en aval à un ou plusieurs modèles de tâches.

Ces règles seront utilisées pour la définition de l'algorithme de transformation présenté dans la suite de ce chapitre.

Exemple de modèle G-DEVS couplé d'un Workflow

Nous avons choisi d'illustrer la création d'un modèle couplé avec les règles de couplage entre les modèles de bases représentant une procédure Workflow. Nous souhaitons représenter dans le formalisme G-DEVS un modèle Workflow composé de trois tâches, la première tâche est enchaînée aux deux suivantes à l'aide d'un connecteur X-Or.

---

[54] Backus-Naur Form est une notation permettant de décrire les règles syntaxiques des langages informatiques. C'est donc un métalangage. BNF est une notation pour des grammaires de type hors contexte (car on définit les termes hors de leur contexte, pour replacer ensuite la définition desdits termes dans ce contexte).

Comme il a été défini précédemment, chaque modèle G-DEVS d'une tâche (Tâche1, Tâche2, Tâche3) doit être associé à un modèle G-DEVS de stock (Stock1, Stock2, Stock3) lui permettant de gérer les arrivées d'Item simultanées ou chevauchées. Les modèles G-DEVS de tâche doivent également être connectés avec un modèle de ressources (Ressource1, Ressource2). Notons que la Ressource2 est connectée à plusieurs tâches. Dans ce cas, l'appel à ces ressources devra être synchronisé, afin d'éviter les ambiguïtés d'allocation.

Enfin, le modèle comprend un modèle G-DEVS de début et un modèle de fin permettant d'initialiser et de récolter les événements sortants afin, éventuellement, de les classer et les analyser.

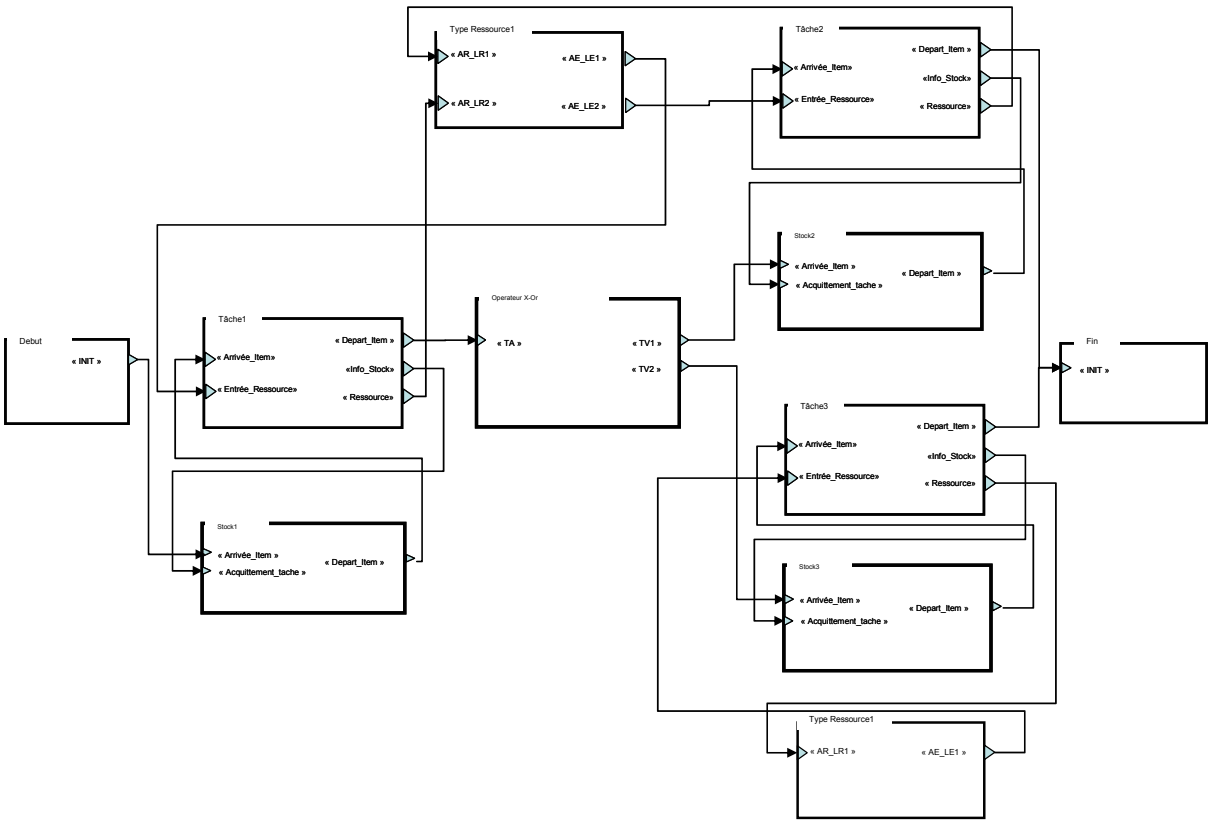


Figure 90 - Modèle couplé G-DEVS d'un Workflow

3.4 Algorithme de transformation

Les éléments basiques composant un Workflow possèdent un modèle atomique G-DEVS équivalent. Ces modèles devront être instanciés en fonction des valeurs des différents paramètres provenant de la structure Workflow.

La génération d'un modèle couplé G-DEVS à partir d'une structure Workflow représentée textuellement doit être automatisée. Pour cela, nous présentons ci-dessous un algorithme de transformation qui génère une structure de modèles G-DEVS couplés à partir d'une structure de modèle Workflow. L'algorithme parcourt la représentation textuelle d'un modèle Workflow et génère à partir de cette structure un modèle couplé G-DEVS. Les structures source et cible sont rappelées ci dessous.

3.4.1 Structure source

Soit une structure de Modèle Workflow **MWF** composée de :

Name, variable contenant le nom du Workflow,

RESS, est la liste des structures ressources du Workflow,

A, est la liste des structures tâches du Workflow,

Ct, est la liste des structures contrôleurs du Workflow,

ARC, est la liste des structures arcs du Workflow,

IT, est la liste des structures Items du Workflow,

ST, est la liste des structures Stocks du Workflow.

3.4.2 Structure cible

La transformation permet d'obtenir une structure de modèle G-DEVS Couplé **MCG-DEVS** composée de :

Name, nom du Modèle couplé G-DEVS

X, ensemble des types des événements externes.

Y, ensemble des types des événements de sortie.

D : ensemble des noms de composants.

Md : modèles G-DEVS.

EIC : couplages d'entrées externes.

EOC : couplages de sorties externes.

IC : couplages internes.

Select : fonction qui définit une priorité entre événements simultanés destinés à des composants différents.

Chaque sous-structure possède récursivement sa propre description structurelle.

3.4.3 Fonction de transformation

Les modèles atomiques G-DEVS « génériques » des différents éléments d'un Workflow `MWF` ainsi que leurs relations de couplage à inclure dans le modèle G-DEVS couplé `MCG-DEVS` sont instanciés par la fonction de transformation présentée ci dessous Figure 91 :

```
Transformation WorkflowG-DEVS (Modèle Workflow MWF)
    Créer un nouveau modèle couplés G-DEVS : MCG-DEVS ;

    Copier MWF.Name dans MCG-DEVS.name

    Parcourir les n éléments de la liste MWF.A :
        Pour chaque WFTa_i ∈ MWF.A, avec 0 < i ≤ n, créer une instance G-DEVSTA_i de modèle
        G-DEVS atomique de Ressource Avec :
            • A partir de WFTa_i.Act (Item) = Item, définir G-DEVSTa_i.λ(En cours
              d'exécution, Item en cours) et G-DEVSTA_i.Act(Item_Cours, Perf_Ress).
            • Pour chaque WFTa_i.R.Resstâche_j // Ensemble des Ressources associée à la tâ-
              che tels que R ⊆ RESS) :
                o Créer un port d'entrée et de sortie « Ressource_i » sur G-DEVSTA_i
                o Inclure le port de sortie associé dans les destinataires de la fonc-
                  tion G-DEVSTA_i.λ(Demande Allocation), dans les entrées de G-DEVSTA_i
                  δext (Attente réponse ressource) et définir la condition sur la va-
                  riable d'état EnsembleRessDisp
```

- Créer les relations de couplage IC entre **G-DEVSTA$_i$** et les modèles de ressources associées et les insérer dans MCG-DEVS.IC

- En fonction de WFTa$_i$.Texec, définir G-DEVSTA$_i$.D(En cours d'execution, Item_encours, Perf_Res), G-DEVSTa$_i$.λ(Allocation Ressource, Item_encours, Perf_Res) et G-DEVSTA$_i$.Act(Item_Cours, Perf_Ress).
- En fonction WFTa$_i$.Sta, créer les relations de couplage IC entre **G-DEVSTA$_i$** et le modèle de Stock associé et les insérer dans MCG-DEVS.IC
- Ajouter **G-DEVSTA$_i$** dans MCG-DEVS.Md
- Ajouter WFTa$_i$.Prt dans MCG-DEVS.Select

Parcourir les n éléments de la liste MWF.CT :

Pour chaque **WFCt$_i$** ∈ MWF.CT, avec 0 < i ≤ n :

Si WFCt$_i$.nom == **So** // c'est un Or Split

- alors créer une instance **G-DEVSSO$_i$** de modèle G-DEVS atomique de Or-Split Avec :
  - Pour chaque élément de l'ensemble WFCt$_i$.o, créer un port de sortie sur G-DEVSSO$_i$, G-DEVSSO$_i$.Itemv.
  - A partir de WFCt$_i$.fxs, définir les conditions associées à la transition, **G-DEVSSO$_i$**.δext(S0)=S1, [Itemv = Item], G-DEVSSO$_i$.δint (S1) = S0, **G-DEVSSO$_i$**.λ(S1) = TVi!Item [Item = Itemv]@(C1 V C2 V ... V Cn) // n nombre de ports de sortie
  - Ajouter **G-DEVSSO$_i$** dans MCG-DEVS.Md

Si WFCt$_i$.nom == **Sx** // c'est un XOr Split

- alors créer une instance **G-DEVSSX$_i$** de modèle G-DEVS atomique de XOr-Split Avec :
  - Pour chaque élément de l'ensemble WFCt$_i$.o, créer un port de sortie sur G-DEVSSX$_i$, G-DEVSSX$_i$.Itemv.
  - A partir de WFCt$_i$.fxs, définir les conditions associées à la transition, **G-DEVSSX$_i$**.δext(S0)=S1, [Itemv = Item], G-DEVSSX$_i$.δint (S1) = S0, **G-DEVSSX$_i$**.λ(S1) = TVi!Item ∀ 1 ≤ i ≤ n [Item = Itemv]@(Ci / 1<i<n) // n nombre de ports de sortie
  - Ajouter **G-DEVSSX$_i$** dans MCG-DEVS.Md

Si WFCt$_i$.nom == **Sa** // c'est un And Split

- alors créer une instance **G-DEVSSA$_i$** de modèle G-DEVS atomique de And-Split Avec :
  - Pour chaque élément de l'ensemble WFCt$_i$.o, créer un port de sortie sur G-DEVSSA$_i$, G-DEVSSA$_i$.Itemv.
  - A partir de WFCt$_i$.fos, définir les ports associés à la transition, **G-DEVSSA$_i$**.δext(S0)=S1, [Itemv = Item], **G-DEVSSA$_i$**.δint (S1) = S0, **G-DEVSSA$_i$**.λ(S1) = TVi!Item ∀ 1 ≤ i ≤ n [Item = Itemv]// n nombre de ports de sortie
  - Ajouter **G-DEVSSA$_i$** dans MCG-DEVS.Md

Si WFCt$_i$.nom == **Jo** // c'est un Or Join

- alors créer une instance **G-DEVSOJ$_i$** de modèle G-DEVS atomique de Or-Join Avec :
  - Pour chaque élément de l'ensemble WFCt$_i$.i, créer un port d'entrée sur G-DEVSOJ$_i$, définition des variables d'état G-DEVSOJ$_i$.nba, G-DEVSOJ$_i$.nbs et G-DEVSOJ$_i$.Tab_Item.
  - A partir de WFCt$_i$.foj, définir **G-DEVSOJ$_i$**.δint,1(S1) = S0 @(*nba == nbs)*, **G-DEVSOJ$_i$**.δint,2(S1) = (S0, [nba++]), **G-DEVSOJ$_i$**.λ(S1) = TV!Item@(C1 V C2 V ... V Cn) conditions sur les ports d'entrée.
  - Ajouter **G-DEVSOJ$_i$** dans MCG-DEVS.Md

Si WFCt$_i$.nom == **Ja** // c'est un And Join

- alors créer une instance **G-DEVSAJ$_i$** de modèle G-DEVS atomique de And-Join Avec :
  - Pour chaque élément de l'ensemble WFCt$_i$.i, créer un port d'entrée sur G-DEVSAJ$_i$, définition des variables d'état G-DEVSAJ$_i$.nba, G-DEVSAJ$_i$.nbs et G-DEVSAJ$_i$.Tab_Item.
  - A partir de WFCt$_i$.faj, définir les fonctions **G-DEVSAJ$_i$**.δint,1(S1) = S0, si *nba < nbs [nba++]*, **G-DEVSAJ$_i$**.λ,1(S1) = Φ, **G-DEVSAJ$_i$**.δint,2(S1) = S0, si *nba == nbs*, **G-DEVSAJ$_i$**.λ,2(S1) = TV!Item [Item = f(Tab_Item)]
  - Ajouter **G-DEVSAJ$_i$** dans MCG-DEVS.Md

Si WFCt$_i$.nom == **Jx** // c'est un Xor Join

- alors créer une instance **G-DEVSXJ$_i$** de modèle G-DEVS atomique de Xor-Join Avec :
  - Pour chaque élément de l'ensemble WFCt$_i$.i, créer un port d'entrée sur **G-DEVSXJ$_i$**, définition des variables d'état **G-DEVSXJ$_i$**.nba, G-DEVSXJ$_i$.nbs et **G-DEVSXJ$_i$**.Tab_Item.
  - A partir de WFCt$_i$.foj, définir **G-DEVSXJ$_i$**.δint,1(S1) = S0 @(*nba == nbs)*, **G-DEVSXJ$_i$**.δint,2(S1) = (S0, [nba++]), **G-DEVSXJ$_i$**.λ(S1) = TV!Item@(C1 V C2 V ... V Cn) conditions sur les ports d'entrée // Diffère de Or join, dans les fonctions de sortie un seul événement est généré
  - Ajouter **G-DEVSXJ$_i$** dans MCG-DEVS.Md

Si WFCt$_i$.nom == **IT** // c'est une Itération

- alors créer une instance **G-DEVSIT$_i$** de modèle G-DEVS atomique d'itération Avec :
  - A partir de WFCt$_i$.fit, définir n = nombre d'iterations, **G-DEVSIT$_i$**.δint(S1) = S0 si 0< *nba* < *n*, **DEVSIT$_i$**.λ(S1) = TV!Item, **DEVSIT-$_i$**.δint1(S1) = S0 si *nba* == *n* et **DEVSIT$_i$**.λ1(S1) = Tfiniteration!Item.
  - Ajouter **DEVSIT$_i$** dans MCDEVS.Md

Parcourir les n éléments de la liste MWF.ARC :

Pour chaque **WFArc$_i$** ∈ MWF.ARC, avec 0 < i ≤ n :

- WFArc$_i$.f (A, CT) créer une relation de couplage EIC, EOC, IC
- Insérer dans **MCG-DEVS**.EIC, **MCG-DEVS**.EOC, **MCG-DEVS**.IC

Parcourir les n éléments de la liste MWF.ITEM :

Pour chaque **WFItem$_i$** ∈ MWF.ST, avec 0 < i ≤ n :

- Créer un événement d'entrée (valeur = **WFItem$_i$** .V, date)
- Insérer cet événement dans la liste des événements d'entrée MCG-DEVS.X.

Parcourir les n éléments de la liste MWF.ST :

Pour chaque **WFSt$_i$** ∈ MWF.ST, avec 0 < i ≤ n, créer une instance **G-DEVSST$_i$** de modèle G-DEVS atomique de stock Avec :

- Récupérer WFSt$_i$.Cap pour définir une valeur max de la variable d'état G-DEVSST$_i$.Stock
- Récupérer WFSt$_i$.Pr pour définir le type de traitement des piles de G-DEVSST$_i$.Stock dans (Stock++ et Stock--)
- Ajouter **G-DEVSST$_i$** dans MCG-DEVS.Md

Parcourir les n éléments de la liste MWF.RESS :

Pour chaque **WFRess$_i$** ∈ MWF.RESS, avec 0 < i ≤ n :

Si **WFRess$_i$** == humain

- alors créer une instance **G-DEVSRESSH$_i$** de modèle G-DEVS atomique de Ressource humaine avec comportement influençable Avec :
  - A partir **WFRess$_i$.Perf et WFRess$_i$.stp**
    - Définir la variable d'état Perf_Ress de **G-DEVSRESSH$_i$**
    - Definir λ (Allocation OK) = DemandeComportement
  - A partir **WFRess$_i$.T** définir un port de sortie sur **G-DEVSRESSH$_i$** pour chaque élément de T
- Ajouter **G-DEVSRESSH$_i$** dans MCG-DEVS.Md

Si **WFRess$_i$** == matérielle

- alors créer une instance **G-DEVSRESSM$_i$** de modèle G-DEVS atomique de Ressource matérielle Avec :
  - A partir **WFRess$_i$.Perf et WFRess$_i$.stp** définir la variable d'état Perf_Ress de **G-DEVSRESSM$_i$**
  - A partir **WFRess$_i$.T** définir un port de sortie sur **G-DEVSRESSM$_i$** pour chaque élément de T
- Ajouter **G-DEVSRESSM$_i$** dans MCG-DEVS.Md

Fin de Transformation WorkflowG-DEVS

Figure 91 – Algorithme de transformation WF - G-DEVS

L'algorithme a permis de générer un modèle MCG-DEVS comportant des modèles G-DEVS atomique équivalent à tous les modèles de Base Workflow MWF. Les relations de cou-

plage entre tous les composants ont été créées à partir des règles de couplage énoncées précédemment.

3.5 Simulation du modèle G-DEVS couplé obtenu

Le modèle couplé G-DEVS obtenu peut ainsi être simulé dans l'environnement LSIS_DME et permettre une analyse du comportement du Workflow au travers de l'étude des états atteints du modèle et des événements de sortie générés.

Nous distinguons dans la simulation deux modes : tout ou partie du Workflow simulé.

Dans le cas d'un Workflow entièrement modélisé et simulé, la simulation pourra être exécutée en intégralité ou par pas. Les résultats seront analysés, soit à chaque étape, soit à la fin de l'exécution. Dans le cas d'une exécution par pas, il sera possible de modifier la partie du modèle qui vient d'être exécutée et relancer la même simulation pour obtenir de nouveaux résultats. Ce mode permet d'anticiper le fonctionnement réel et ainsi d'éviter des erreurs lors de la construction d'un processus réel. Dans ce cas, l'intégralité des éléments du Workflow est simulée.

Dans le cas où elle s'intègre à une exécution reliée à des processus réels (donc en temps réel), elle sera exécutée par étape. A chaque étape, la procédure simulée est stoppée et déclenche l'intervention d'acteurs humains (leurs demandant de prendre une décision) ou d'application invoquées. Après obtention des résultats externes, les acteurs du Workflow analysent les résultats au travers de l'interface de monitoring et choisissent de continuer la procédure sur le même modèle, ou peuvent modifier le modèle pour la suite et relancer l'exécution pour obtenir les résultats suivants.

4 Définition d'un environnement Workflow compatible HLA

Un des enjeux actuel des environnements de gestion de procédures est la capacité de ces environnements à pouvoir exécuter des modèles distribués et à pouvoir s'interfacer avec des applications hétérogènes. Une idée consiste donc à permettre à l'environnement Workflow de s'exécuter à partir de plusieurs ordinateurs distants et de pouvoir se coupler à des applications de monitoring, des applications utilisées ou des interfaces homme/machine réparties commandées par une des tâches du Workflow, ou même interopérer avec d'autres Workflow.

4.1 Travaux précédents

Nous avons proposé dans [ZACHAREWICZ 02] une première architecture distribuée de gestion de Workflow basée sur les réseaux de Petri. La communication des différents composants distribués de cette solution est assurée grâce à leurs conformités à la norme HLA.

Cette structure comprenait uniquement les outils de définition de procédures connectés sur l'interface 1, c'est-à-dire la partie modélisation et simulation de l'environnement Workflow, et ne prenait pas en considération les autres interactions. Néanmoins, cette structure a permis de donner une base de réflexion sur les structures à mettre en œuvre dans le cadre de

l'élaboration d'un environnement Workflow. Nous présentons ci-dessous la Figure 92 extraite de [ZACHAREWICZ 02] illustrant la composition d'un modèle de processus Workflow réparti communiquant au travers du RTI.

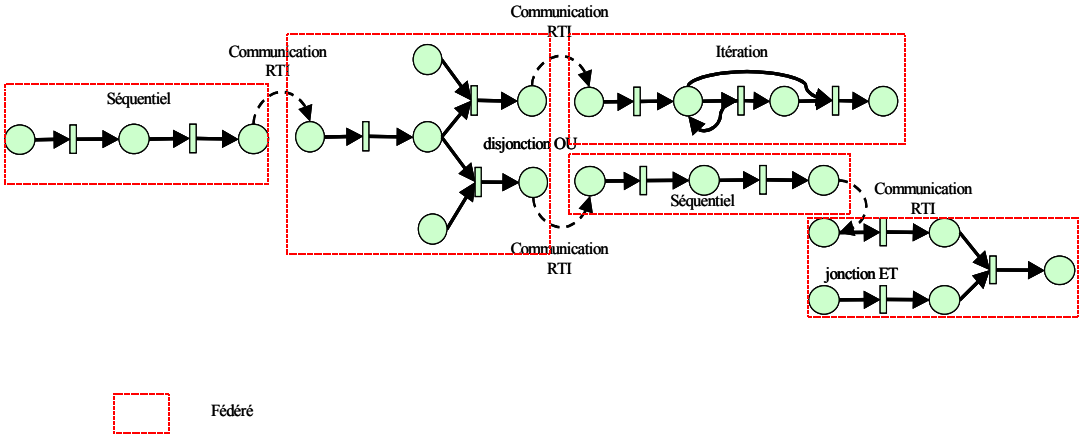


Figure 92 - Réseau de Petri associé avec communication avec le RTI

Plus tard, des travaux généralisant la distribution des composants de l'architecture d'un Workflow ont été proposés par [BUBAK 03]. Cette architecture repose sur la notion de Grid [55]. Dans la Figure 93, [ZAJAC 03] proposent une architecture de services de Grid qui coopèrent pour initialiser et lancer des applications interactives distribuées. L'utilisateur exécute le service d'Exécuteur de Flux et lui fournit le code d'application qui correspond aux données à manipuler (déjà compilé et dynamiquement lié avec la bibliothèque du RTI HLA). Lorsque les informations nécessaires sont fournies, l'Exécuteur de Flux transfère le code d'application aux ressources et demande aux composants d'Initialisation de Flux Locaux concernés de démarrer tous les services locaux qui seront ensuite contrôlés pendant leurs exécutions. Enfin il planifie la migration du code vers les autres fédérés HLA locaux.

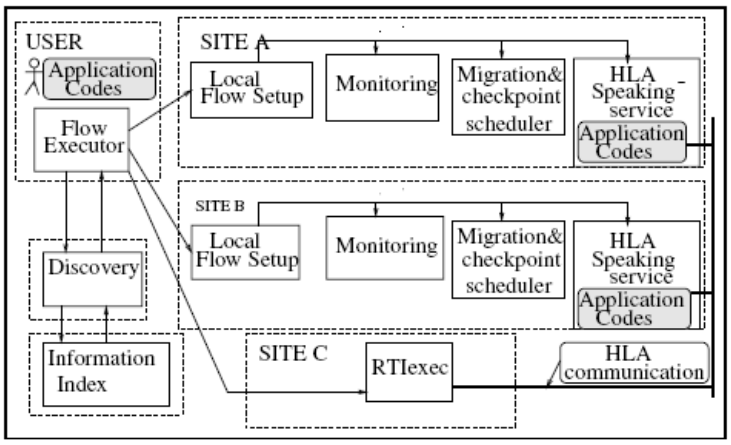


Figure 93 - Environnement de gestion de flux compatible « HLA »

[55] Une grille informatique est une infrastructure virtuelle constituée d'un ensemble coordonné de ressources informatiques partagées, distribuées, hétérogènes, externalisées et sans administration centralisée.

4.2 Modèle de référence Workflow compatible HLA

En nous appuyant sur la synthèse des architectures présentées dans le § 4.1., nous avons proposé dans [ZACHAREWICZ 06b] une nouvelle architecture d'environnement Workflow distribuée. Cette nouvelle architecture se base sur le modèle de référence proposé par la WfMC (cf. Figure 18), qui présente les différentes zones avec lesquelles le noyau d'exécution doit être interfacé. Les interfaces sont assurées, dans cette structure, par un mécanisme de communication entre les composants distants. Ce type de communication est rendu possible par la conformation des composants à la norme HLA. La Figure 94 présente ce nouveau modèle de référence Workflow basé sur G-DEVS / HLA.

Dans ce modèle, les parties modélisation & simulation sont basées sur le formalisme G-DEVS et plongées dans un contexte compatible HLA. Les modèles sont exécutés de façon distribuée en s'appuyant sur l'environnement G-DEVS/HLA introduit dans le chapitre précèdent. Ils communiquent avec les autres zones de l'environnement Workflow au travers de l'interface 1. La communication entre les modèles en simulation est assurée par des interactions HLA comme cela a été défini au chapitre précédent. Néanmoins, Il est nécessaire de créer de nouveaux objets HLA pour établir la communication avec les autres zones telles que le monitoring, les applications invoquées et les applications faisant intervenir un utilisateur humain.

L'interaction 2 avec un utilisateur humain peut être assurée au travers d'une interface graphique définie sous forme d'application web. Ces aplications d'IHM[56], appelées pendant l'exécution du Workflow, sont intégrées au fonctionnement global en se conformant à la norme HLA ; plus concrètement en partageant des objets et des interactions au sein de la fédération.

L'interaction 3 assure le lien entre l'environnement et des applications invoquées, ces applications sont invoquées automatiquement par le moteur de Workflow. Afin de pouvoir interpréter les informations provenant du Workflow et envoyer des réponses dans un format commun aux composants distribués, ces applications devront donc également être conformes à la norme HLA, elles définiront pour cela des objets partagés et utiliseront des services de publication et de souscription fournis par le RTI.

De plus, le coordinateur racine distribué de la structure de simulation G-DEVS distribuée a été remplacé dans le § 3.3. du chapitre 2 par le RTI. Nous proposons d'extrapoler cette solution au remplacement du moteur Workflow par le RTI. En effet, le RTI est capable d'assurer les fonctions définies pour un moteur d'exécution du Workflow, en assurant le routage des informations à échanger par la définition d'objets et d'interactions partagées par la fédération.

Enfin, nous pouvons envisager l'interaction 4 avec les autres Workflow, c'est-à-dire l'interaction avec d'autres fédérations. Cette interaction pourra être assurée par des fédérés pont [BRÉHOLÉE 03] entre les différentes fédérations d'environnement Workflow, assurant ainsi la sécurité et la confidentialité dans les informations échangées.

[56] Interface Homme-Machine

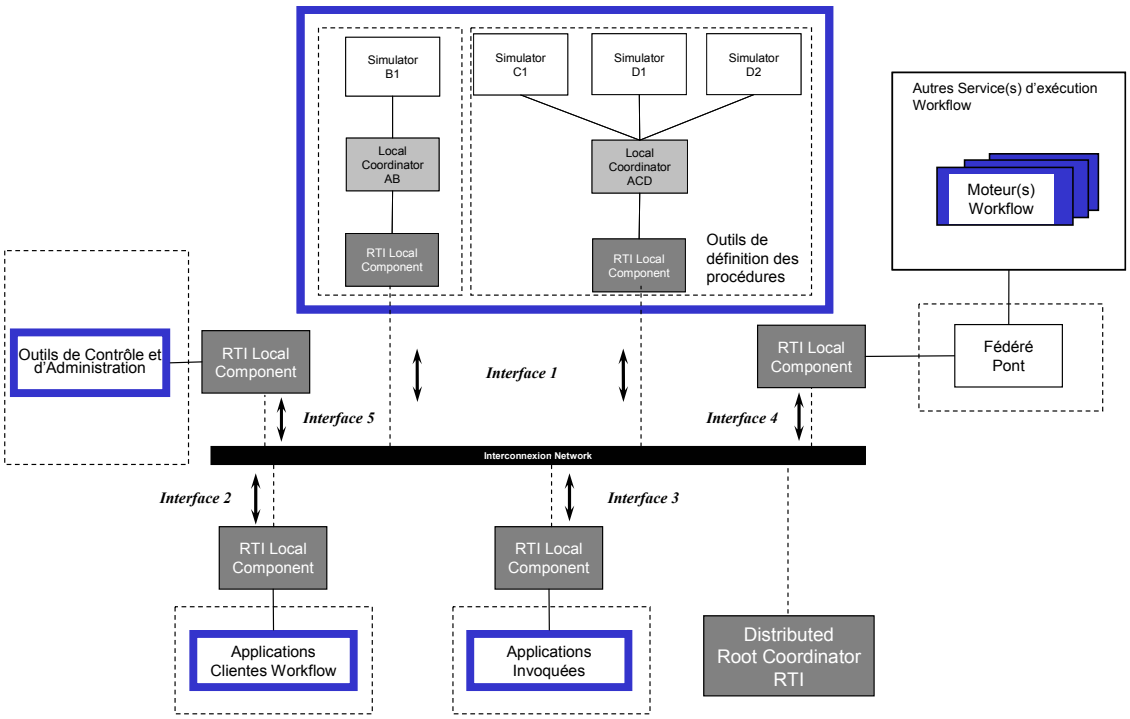


Figure 94 - Modèle de référence Workflow compatible HLA

4.3 Création des tables de FOM de la fédération Workflow GDEVS HLA

En complément des tables d'interactions définies au chapitre précédent pour la fédération HLA, il est nécessaire de définir des tables de classes d'objets afin d'assurer la communication entre les modules de M&S et les autres zones d'un environnement Workflow. Ces classes génériques sont présentées Figure 95.

Object Class Structure Table			
HLA object Root (N)	Interface (S)	Interface 1 (PS)	Contenu Evénements (PS)
			Table d'etats actuels des modèles (PS)
			Information Début / Fin de simulation (PS)
		Interface 2 (S)	Modification du contenu d'Item pour routage (PS)
			Modification du contenu d'Item par opération non gérée par la Workflow (PS)
		Interface 3 (S)	Information provenant de l'aplication invoquée (PS)
		Interface 4 (S)	Information vers le fédéré pont appartennant aux deux Fédération de Workflow (PS)
		Interface 5 (S)	Information d'administration en modification du modèle (PS)
			Information d'administration en paramétrage des applications invoquées (PS)

Figure 95 - Table d'objet du FOM d'un Workflow

La classe d'objet Interface 1 assure la diffusion des informations provenant de la simulation du modèle. En effet, il a été nécessaire de rajouter des classes pour diffuser les informations

relatives aux états atteints du modèle, en détaillant les valeurs des variables d'états ainsi qu'une liste présentant l'historique persistante des événements échangés dans le modèle. Ces données sont ainsi analysées avec les activités de Monitoring et utilisées par les applications invoquées et/ou par les applications clientes.

La classe d'objet Interface 2 et 3 assure la diffusion des informations provenant des applications qui interviennent dans la procédure Workflow. Elle assure, en fait, la récupération des informations après traitement par les applications et leur diffusion au sein de la fédération pour les autres fédérés intéressés (Typiquement les fédérés de monitoring et de M&S).

La classe d'objet Interface 4 assure la communication avec les autres Workflow. Pour des raisons de confidentialité et pour simplifier la définition des fédérations Workflow, nous proposons d'utiliser la notion de fédéré pont introduite par [BRÉHOLÉE 03]. Cette solution consiste à définir un fédéré à la frontière de deux fédérations. Ce fédéré publie et souscrit à des objets et aux interactions des deux fédérations. La définition des FOM de chaque fédération est donc statique et, la confidentialité est accrue par le fait que chaque fédération ne perçoit des autres fédérations que les informations partagées au travers du fédéré pont.

La classe d'objet Interface 5 assure la récupération des informations de « monitoring » provenant des autres zones de l'environnement Workflow. Ces informations seront affichées au travers d'une interface graphique utilisateur. Cette interface offre également la possibilité d'assurer des fonctions de monitoring sur la procédure un cours, dans le cas où l'utilisateur effectue une action sur le Workflow en cours. Les informations seront diffusées à l'aide de la classe d'information d'administration, à laquelle auront souscrit les zones de l'environnement sujettes à modification. Concrètement, l'utilisateur peut souhaiter modifier la procédure suite à un résultat de simulation non satisfaisant. Dans ce cas, il saisira les modifications sur l'interface d'administration, les informations provenant de cette interface seront stockées dans l'objet « information d'administration » contenant la structure du modèle. La zone de M&S, principalement, ayant souscrit à cet objet, recevra ces informations et pourra modifier son modèle en conséquence. Enfin, il doit être possible de modifier et paramétrer les applications invoquées ou clientes, c'est pourquoi nous avons défini une classe de modification des applications, à laquelle souscrivent les différents fédérés des programmes invoqués ou externes.

5 Application à un Workflow d'entreprise

Pour illustrer l'utilisation de l'environnement Workflow G-DEVS/HLA, nous avons choisi de modéliser une partie du processus industriel mis en œuvre par la société STMicroelectronics sur son site de Rousset.

La phase de production d'un circuit intégré peut être divisée en deux étapes. La première, la fabrication des wafers, est un processus de fabrication extrêmement sophistiqué et complexe. La seconde, l'assemblage, est un processus très précis et automatisé. Ces deux phases sont plus communément connues respectivement sous le nom de Front-end et Back-End. La première peut être définie comme la première étape de fabrication et de tests (First manufac-

turing step and test), et la seconde comme l'installation qui fournit l'assemblage et le package final du produit.

Dans ce travail, nous nous intéressons à la première étape en étudiant les principales tâches de fabrication des puces, et en détaillant par la suite les procédures de tests appliqués à la fin de cette fabrication.

Le Workflow modélisé sera exécuté en mode « analyse », il ne sera donc pas couplé en temps réel avec le processus réel. Il sert dans notre cas à valider par la simulation, la mise en place de nouvelles procédures, en anticipant les situations d'erreurs et en les corrigeant en prévention.

5.1 Cahier des charges

La société STMicroelectronics nous a proposé des schémas de leurs flux de production nommés en terme métier « routes productives ». Il s'agit d'une description sous forme de tableaux, de routes et des scripts d'opérations les composant. Chaque **route** contient un ensemble d'opérations. Une opération est détaillée par une représentation graphique nommée **Script**.

Début : Bloc de début dans la représentation d'une route, correspondant au début de la production au niveau de la route. Ce bloc est également présent en début du script représentant une opération.

Fin : Bloc de fin dans la représentation d'une route, correspondant à la fin de la production au niveau de la route. Ce bloc est également présent en fin du script représentant une opération.

Opération : Bloc de travail sur une plaquette dans l'une des zones de l'usine (e.g. diffusion, photolithographie, gravure, implantation, etc.). Chaque opération contient, au minimum, trois blocs nommés en terme métier steps qui sont présents dans tous les scripts d'opération mais restent habituellement sous-entendus (10 : Start, 20 : Move in, 9999 : Move out) ainsi qu'un ensemble d'événements ou « Event ».

Step 10; **Start** : première procédure de toute opération, indiquant le début de la procédure considérée.

Step 20; Move IN : indique la mise en place du lot dans la procédure de l'opération.

Step 9999 : Move Out : indique la sortie physique du lot de l'opération**.**

Event X avec Equipement Y : contient un numéro de Step, Event, Description, Paramètres : pour les Event MSL et MSE (cpklag, limites (UCL, LCL, USL, LSL), Target, Unité, Wafer, Sit, Chart, Ocap). Pour les Events PRP : recipe. Pour les Event PRM : recipe

Il est possible de représenter, pour donner une vision plus concrète des termes métiers énumérés ci-dessus, les routes et les scripts par une notation graphique informelle. Un exemple générique est donné dans la Figure 96.

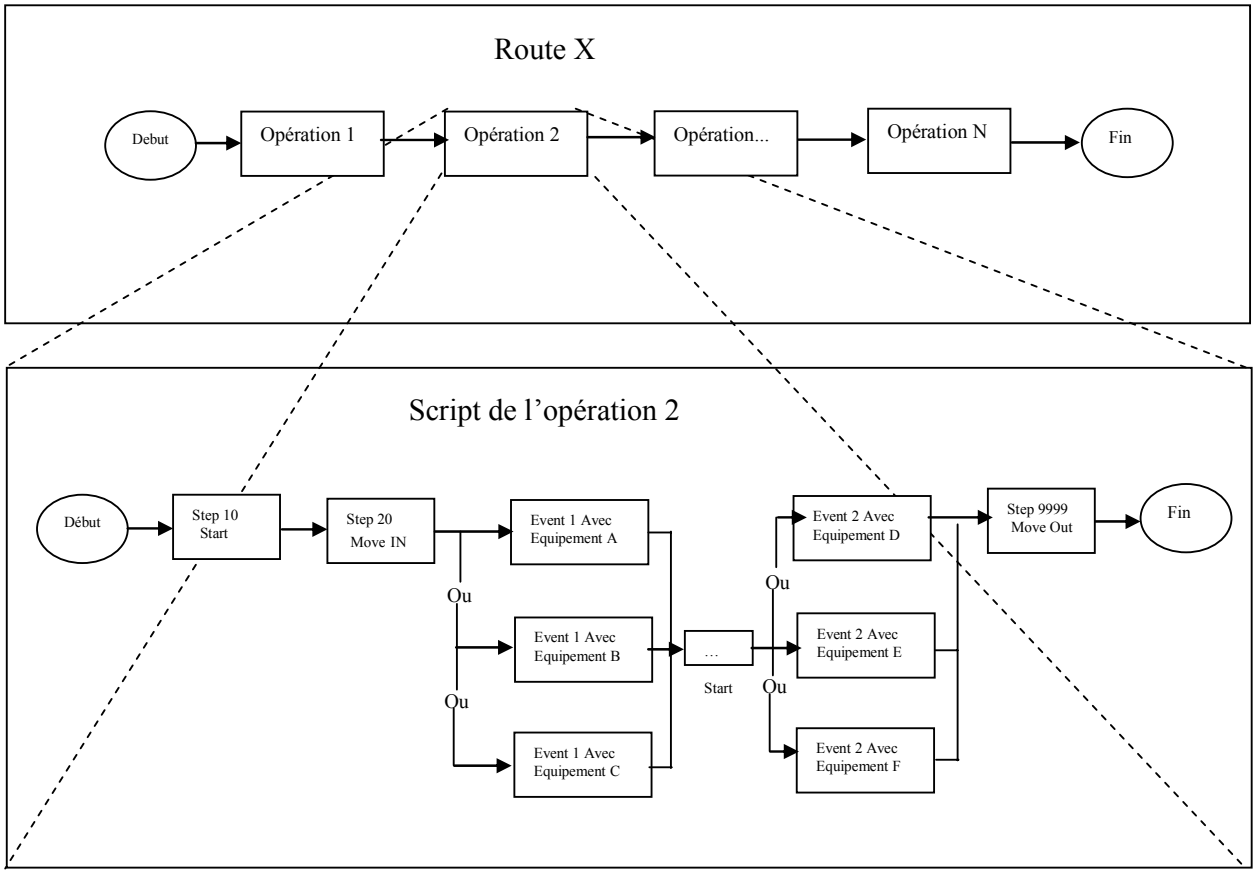


Figure 96 - Route Générique STMicroelectronics

5.2 Exemple de la Route H80SH27F attachée au produit FDHB2AAG-03

Nous présentons ici la route H80SH27F permettant de marquer une puce electronique à l'aide d'un laser et de la traiter par oxydation. La société nous a fournis des tables (l'extrait relatif à H80SH27F est présenté Figure 97) présentant les différents équipements autorisés à exécuter un « event » ainsi que les données limites de spécification et de contrôle.

oper	oper desc	step	event	recipe	job	event desc
1010	Laser Marking	3010	PRPMKL0047B	MKL0047B		Laser Marking
1015	Clean 0 level oxidatio	3010	PRPN1RCA01	N1RCA01A		RCA Clean Classe 1
		3010	PRPN1RCA01	N1RCA01A		RCA Clean Classe 1
1020	0 level oxidation	3010	PRPOXIDE01	OXIDE01		OXI 12nm / 925C / 19mn / dry
		3010	PRPOXIDE01	OXIDE01		OXI 12nm / 925C / 19mn / dry
		5611	PRMOXIDE01E	DF-NPOX01		Thickness Measurement
		5611	PRMOXIDE01E	DF-NPOX01		Thickness Measurement
		5611	PRMOXIDE01E	DF-NPOX01		Thickness Measurement
		5711	MSLOXIDE01E			THICK. RECORDING (OXIDE01) PRODUCT
		5711	MSLOXIDE01E			THICK. RECORDING (OXIDE01) PRODUCT
		5711	MSLOXIDE01E			THICK. RECORDING (OXIDE01) PRODUCT
		5711	MSLOXIDE01E			THICK. RECORDING (OXIDE01) PRODUCT
		5711	MSLOXIDE01E			THICK. RECORDING (OXIDE01) PRODUCT
		5711	MSLOXIDE01E			THICK. RECORDING (OXIDE01) PRODUCT

param	cpk_flag	lsl	usl	lcl	ucl	target	unite	waf	sit	chart	ocap	eqpt	eqpt type
												LAS	LUMLM
												WDC	DNS_CFC821L
												WDC	DNS_CWS820L
												FOX	TELOXCL1
												FPG	ASMPG400
												THM	KLAFTASETF5
												THM	METFTRUDOLPH
												THM	THEFTOPTIP
EOXZL	N	106	130	113.5	122.5	118	ANG	2	9	SCP$ROBERTS	E012830	FOX	TELOXCL1
EOXZL	N	106	130	113.5	122.5	118	ANG	2	9	SCP$ROBERTS	E018893	FPG	ASMPG400
EOXZL	N			0	4	2	ANG	2	9	SCP$XDELTAR	E018893	FPG	ASMPG400
EOXZL	N			0	4	2	ANG	2	9	SCP$XDELTAR	E012830	FOX	TELOXCL1
EOXZL	N			0	3.4000000954	1.7000000477	ANG	2	9	SCP$XDELTAW	E018893	FPG	ASMPG400
EOXZL	N			0	3.4000000954	1.7000000477	ANG	2	9	SCP$XDELTAW	E012830	FOX	TELOXCL1

Figure 97 - Route H80SH27F attachée au produit FDHB2AAG-03.

Le parcours des tableaux fournis par la société STMicroelectronics pour la route H80SH27F, permet de définir les différents composants du Workflow.

En l'occurrence, les différentes couleurs du tableau isolent les opérations 1010, 1015 et 1020, auxquelles nous devrons, par la suite, ajouter des opérations implicites telles que « … » (sans nom) et 9200 qui sont implicites et qui doivent être ajoutées à la modélisation. Les différentes ressources intervenant dans cette route sont présentées dans la colonne « eqpt type ». Enfin, les items (wafers électroniques) en circulation sont de type FDHB2AAG-03.

Nous allons donner, par la suite, une représentation dans la forme graphique (cf. Figure 96) puis textuelle (cf. § 3.2.3.) du Workflow de la route H80SH27F.

5.2.1 Représentation du modèle à l'aide de l'outil de modélisation graphique LSIS_WME

Une modélisation de Workflow de la route H80SH27F a été proposée en utilisant l'éditeur de modèle de Workflow LSIS_WME (voir ci-après) développé par le LSIS. L'objectif de cet outil logiciel d'édition est de simplifier et de sécuriser la création d'une spécification Workflow.

5.2.1.1 Présentation de l'éditeur de Workflow LSIS_WME

Cet éditeur représente un flux de travail souhaité sous forme graphique en respectant des règles grammaticales basées sur des prémisses issus de la WFMC afin d'en assurer la représentation explicite.

Ce logiciel implémente les fonctionnalités suivantes :

- Création d'une spécification Workflow hiérarchisée graphique.
- Gestion des différents contrôleurs de flux :
- And Join, And Split, Or Join, Or Split, Xor Join et Xor Split
- Edition de tous les paramètres d'une tâche :
- Nom, Type, Priorité, Description, Action, Ressources
- Vérification syntaxique en ligne du Workflow :

- Vérification des connexions interdites, et de la structure explicite du Workflow.
- Fonctionnalités en développement :
- Gestion avancée des ressources.
- Création et gestion de triggers.
- Gestion des multi-instances et de la hiérarchie du flux.
- Editions avancées des transitions du flux.
- Exports vers des formats différents.

Le logiciel LSIS_WME est disponible actuellement en version 1.101.B, cette version beta, présente d'ores et déjà les orientations majeures suivies. Cependant, des évolutions majeures et nécessaires sont en développement afin de rendre ce logiciel complet.

5.2.1.2 Représentation graphique de haut niveau de la route H80SH27F

La Figure 98 représente le modèle Workflow de haut niveau de la route **H80SH27F** modélisé sur LSIS_WME. Ce modèle est composé de plusieurs tâches qui sont des tâches de base ou des sous-Workflow dont la structure n'est pas représentée à ce niveau sur la figure.

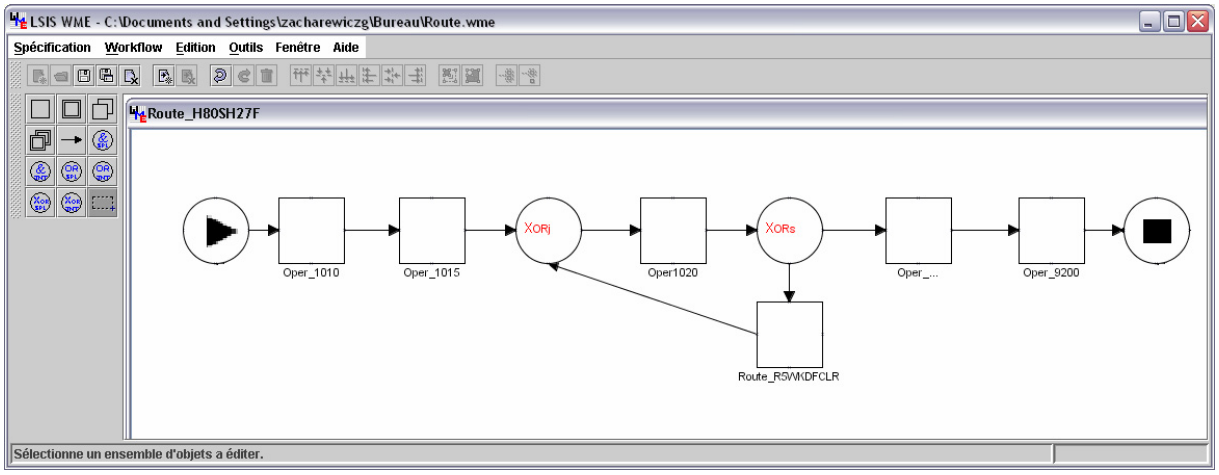


Figure 98 - Modèle Workflow de la route H80SH27F STMicroelectronics

Le contrôleur XOR-Split (XORs) constitue un test sur les limites de spécification et de contrôle. En effet, à la sortie du poste de travail (Oper 1020), le lot est, soit dirigé vers l'opération suivante, soit vers une route de *rework*[57] (Route_RWKDFCL1R).

5.2.2 Représentation textuelle de la route H80SH27F

A partir du tableau fourni par la société STMicroelectronics, ou à partir de la sortie de l'éditeur graphique, nous proposons une représentation textuelle de la route, définie par la structure algébrique suivante.

W: (Name, RESS, A, ARC, IT, ST).

Name= 'Route_H80SH27F'.

[57] Retravailler, refaire

RESS= {(LAS01, n1), (WDC03, n2), (WDC04, n3), (FOX01, n4), (FPG01, n5), (THM06, n6), (THM05, n7), (THM04, n8), (FOX01, n9), (FPG01, n10), (…,…)}.

En détail chaque ressource est de la forme : LAS01= (WDC03, LUMLM, Perf_LAS01, {Oper_1010, Oper_…}).

A= {Oper_1010, Oper_1015, Oper_1020, ..., Route_RWKDFCL1R, Oper_9200}.

Ct= {(Xor-Join1, Xor-Join), (Xor-Split1, Xor-Split), …}.

ARC= {(Debut_Flux, Oper_1010), (Oper_1010, Oper_1015), (Oper1015, Xor-Join1), (Xor-Join1, Oper_1020), (Oper_1020, Xor-Split1), (Xor-Split1, Route_RWKDFCL1R), Route_RWKDFCL1R, Xor-Join1), (Xor-Split1, Oper_…), …, (Oper_9200, Fin_Flux)}.

IT= {Ident_Lot}. *Ident_Lot*: un lot de 25 wafer; chaque wafer est défini par un ensemble de paramètres.

ST= {Stock_A, Stock_B, Stock_C}.

5.2.3 Représentation des sous-modèles

5.2.3.1 Modèle Workflow de la route de rework RWKDFCL1R

La route de rework Route_RWKDFCL1R permet d'effectuer une opération de modification des opérations contenues dans la route considérée.

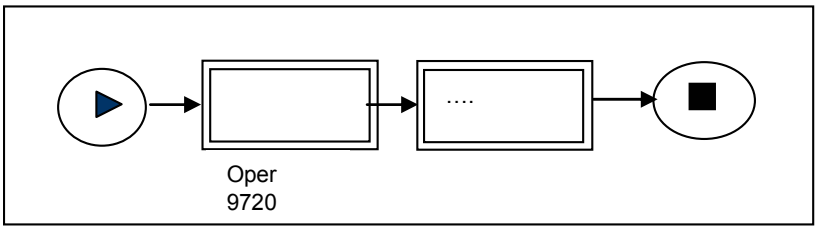


Figure 99 - Modèle Workflow de la route de rework RWKDFCL1R

Cette sous-route comporte les opérations 972 et « … », permettant d'assurer une modification du processus sur la route principale. L'objectif est de pouvoir corriger des Wafers défectueux.

5.2.3.2 Modèle Workflow de la tâche composite Oper_1010

Elle est composée par un seul *Event* nommé 3010, caractérisé par le PRP 'PRPMKL0047B' et sa « recette » 'MKL0047B'. L'équipement utilisable est 'LUMLM' (famille 'LAS'). Il s'ajoute à cela les *Event* implicites 10, 20 et 9999.

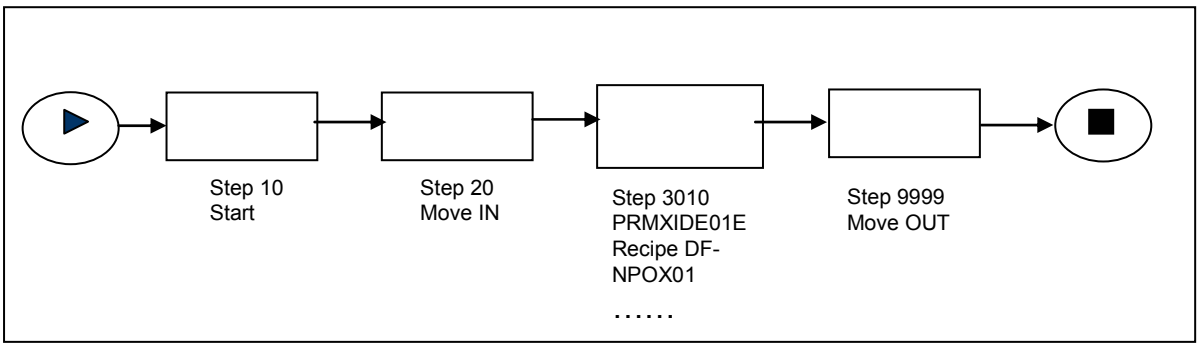


Figure 100 - Modèle Workflow du Script OF Markl 03 (opération 1010 Laser Marking)

W: (Name, RESS, A, EN, IT, ST).

Name = 'Script_OF_MARKL_03'.

RESS = {(LAS01, n1)}

LAS01= (LAS01, LUMLM, Perf_LAS01, {Script_OF_MARKL_03}).

A = {Step_10, Step_20, Step_3010, Step_9999}.

Ct = {ϕ}.

ARC = {(Debut_Flux, Step_10), (Step_10, Step_20), (Step_20, Step_3010), (Step_3010, Fin_Flux)}.

IT = {Ident_Lot}.

ST = {Stock_A}.

Stock_A= ('Stock des wafer de lot ayant besoin de l'équipement LAS01', Cap1, FIFO).

Description des tâches atomiques composants le modèles Workflow (Tâche composite) Script_OF_MARKL_03

Step_3010 = (Step_3010, 'un process sur la plaquette décrit dans l'event PRPMKL0047B', 'DF-NPOX01', {LAS01}, Temps_Exec_PRPMKL0047B, Et, Prt).

L'EVENT PRPMKL0047B est décrit par une liste d'attributs.

Name = 'PRPMKL0047B'.

Recipe = 'MKL0047B'.

Eventdescription = 'Laser Marking'.

Ent = 'LAS01'.

Equipment_Type = 'LUMLM'.

5.2.3.3 Modèle Workflow de la tâche composite Oper_1015

La tâche Oper_1015 est elle même un Workflow. Cette tâche permet d'effectuer une opération de nettoyage sur le Wafer en cours de production.

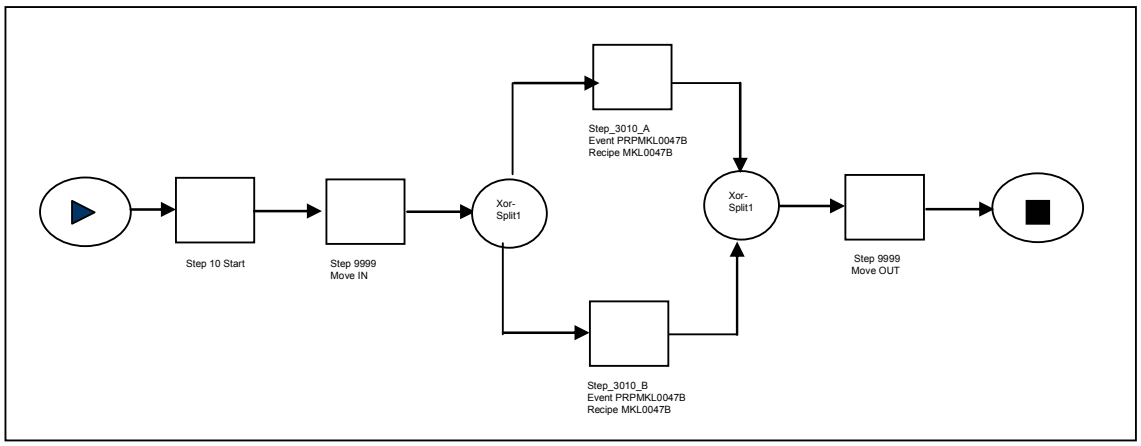


Figure 101 - Modèle Workflow de Script of Clean 06 (opération 1015)

W: (Name, RESS, A, EN, IT, ST).

Name = Script_OF_CLEAN_06.

RESS = {(WDC03, n2), (WDC04, n3)}

WDC03 = (WDC03, DNS_CFC821L, Perf_WDC03, {Step_3010_A}).

WDC04 = (WDC04, DNS_CWS820L, Perf_WDC04, {Step_3010_B}).

A = {Step_10, Step_20, Step_3010_A, Step_3010_B, Step_9999}.

Ct = {(Xor_Split1, Xor-Split), (Xor-Join1, Xor-Join)}.

ARC = {(Debut_Flux, Step_10), (Step_10, Step_20), (Step_20, Xor-Split1), (Xor-Split1, Step_3010_A), (Xor-Split1, Step_3010_B), (Step_3010_A, Xor-Join1), (Step_3010_B, Xor-Join1), (Xor-Join1, Step_9999), (Step_9999, Fin_Flux)}.

IT = {ident_Lot}.

ST = {Stock_B}.

5.2.3.4 Modèle Workflow de la tâche composite Oper_1020

La tâche Oper_1015 est elle-même un Workflow qui permet d'apporter une valeur ajoutée sur le Wafer. Le flux du Workflow partant du step 20 (accusé de réception du lot) est adressé vers le premier Event du script de l'opération (i.e. 3010) qui peut être accompli avec deux équipements différents ('FOX01' de la famille 'TELOXCL1') ou ('FPG01' de la famille 'ASMPG400'). Le choix entre les deux équipements est assuré par un logiciel spécifique qui gère toutes les ressources et qui reçoit des informations déterminant la machine qui va travailler sur le lot. Ce choix est représenté par le premier XOR Split dont les sorties définissent deux chemins possibles pour un lot. Ces flux sont ensuite dirigés vers l'Event suivant, quelque soit l'équipement utilisé, après une jonction XOR Join, située entre la fin du step 3010 et le début du step 5611.

On observe le même principe pour le deuxième XOR Split, qui permet de choisir entre trois équipements utilisables pour l'event 5611. Dès que le step 5611 est achevé, les flux sont une nouvelle fois concentrés par un XOR Join situé entre la fin du step 5611 et le début du step 5711. La dernière disjonction de chemin, en amont du step 5711, est due à un choix de vérificateurs de mesures sur les Wafers dépendant de la machine ayant précédemment effec-

tué le PRP (event 3010). Enfin, le dernier step indique la fin de script de l'opération, (i.e. step 9999).

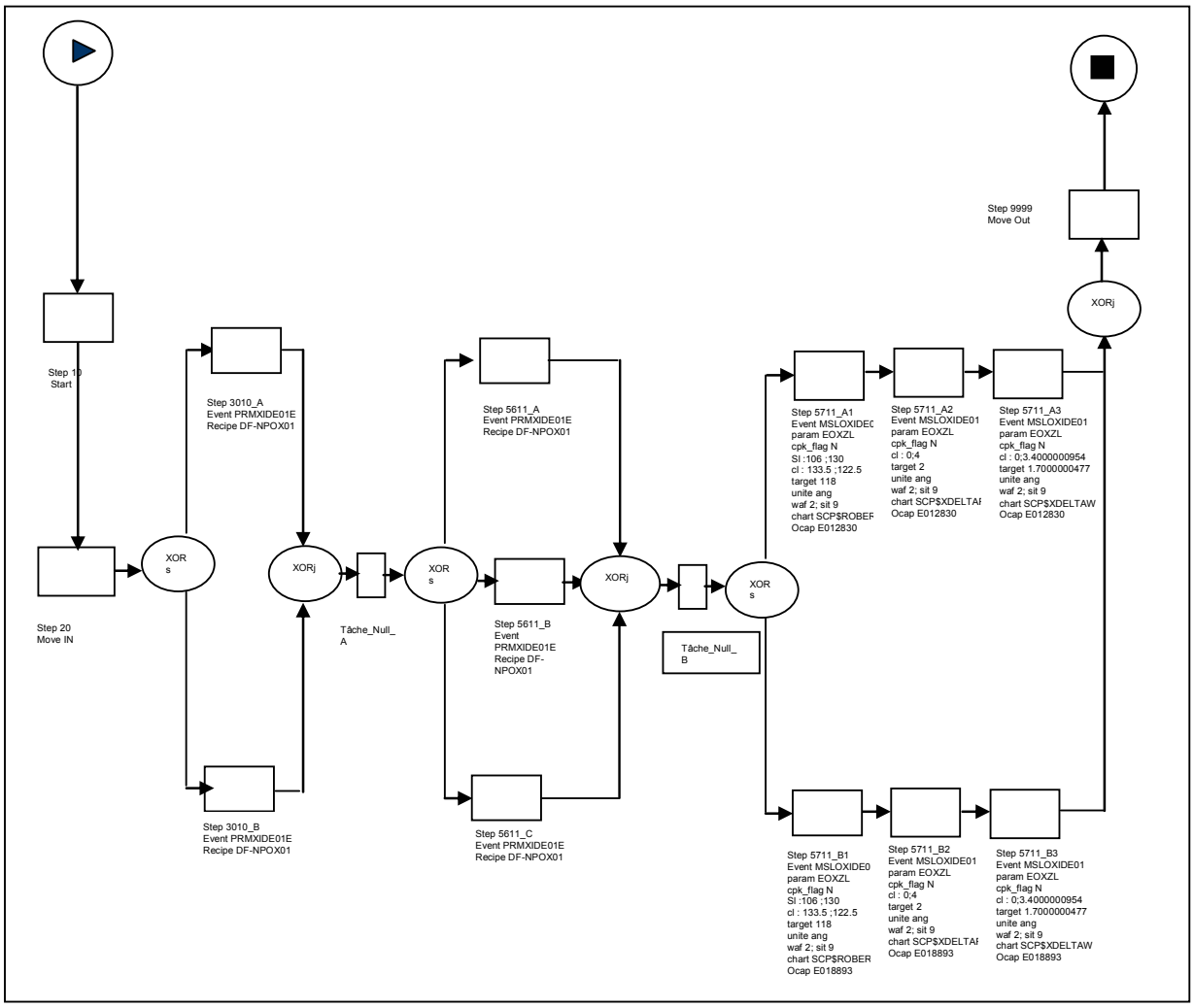


Figure 102 - Modèle Workflow du Script_XXX (opération 1020)

Name = 'Script_XXX'.

RESS = {(FOX01, n4), (FPG01, n5), (THM06, n6), (THM05, n7), (THM04, n8)}.
FOX01= (FOX01, TELOXCL1, Perf_FOX01, **Script_XXX**}).

A = {Step_10, Step_20, Step_3010_A, Step_3010_B, Step_5611_A, Step_5611_B, Step_5611_C, Step_5711_A1, Step_5711_A2, Step_5711_A3, Step_5711_B1, Step_5711_B2, Step_5711_B3, Step_9999}.

Ct = **{(**Xor-Split_1, Xor-Split), (Xor-Split_2, Xor-Split), (Xor-Split_3, Xor-Split), (Xor-Join_1, Xor-Join), (Xor-Join_2, Xor-Join), (Xor-Join_3, Xor-Join)**}.**

ARC = {(Debut_Flux, Step_20), (Step_20, Xor-Split_1), (Xor-Split_1, Step_3010_A), Xor-Split_1, Step_3010_B), (Step_3010_A, Xor-Join_1), (Step_3010_B, Xor-Join_1), (Xor-Join_1, Tâche_Null_A), (Tâche_Null_A, Xor-Split_2), (Xor-Split_2, Step_5611_A), (Xor_Split2, Step_5611_B), (Xor-Split_2, Step_5611_C), Step_5611_A, Xor-Join_2), (Step_5611_B, Xor-Join_2), (Step_5611_C, Xor-Join_2), (Xor-Join_2, Tâche_Null_B), (Tâche_Null_B, Xor-Split_3), (Xor-Split_3, Step_5711_A1), (Xor-Split_3, Step_5611_B1),

(Step_5711_A1, Step_5711_A2), (Step_5711_A2, Step_5711_A3), (Step_5711_B1, Step_5711_B2), (Step_5711_B2, Step_5711_B3), (Step_5711_A3, Xor-Join_3), (Step_5711_B3, Xor-Join_3), (Xor-Join_3, Step_9999), (Step_9999, Fin_Flux)}.

IT = {Ident_Lot}.

ST = {Stock_C1, Stock_C2}.

5.3 Transformation du Workflow en modèle G-DEVS

Le modèle Workflow est ensuite transformé en modèle couplé G-DEVS en appliquant les règles de transformation définies précédemment. La Figure 103 présente le modèle couplé G-DEVS de la route H80SH27F.

Ce modèle simulable permet d'obtenir des informations sur l'efficacité de la route mise en œuvre. En pratique, l'analyse des événements échangés et des états atteints des modèles permet, en particulier, de déterminer un bon acheminement des produits, d'identifier la constitution de goulots d'étranglement, ainsi que la détection des branches d'une route éventuellement inutilisées.

Cette simulation permet donc d'anticiper un changement effectif d'une route et d'en mesurer les coûts par simulation, évitant ainsi les erreurs de la mise en place réelle de mauvaises procédures.

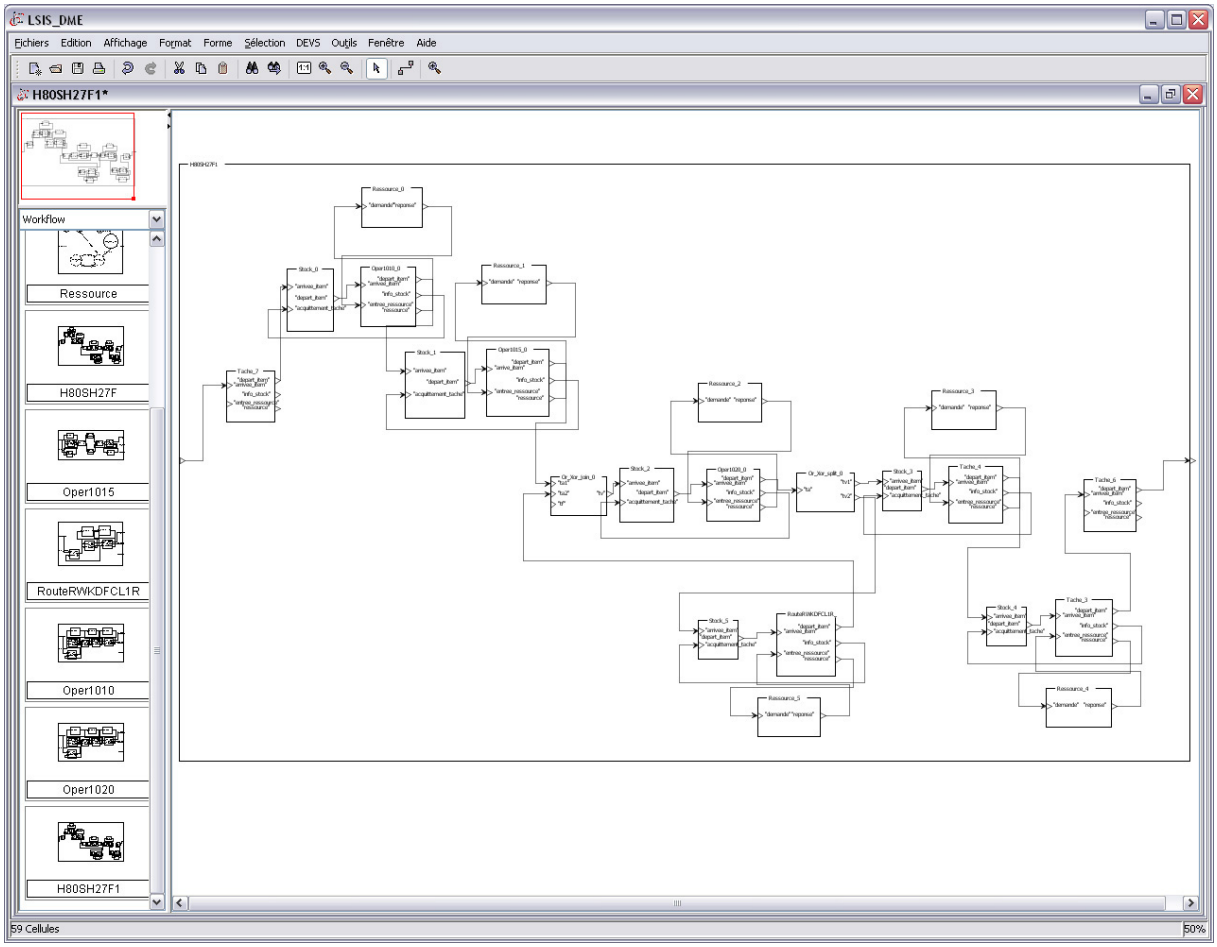


Figure 103 - Modèle G-DEVS couplé de la route H80SH27F sur LSIS_DME

5.3.1 Résultats de simulation de la route H80SH27F

La modélisation de la route H80SH27F a été effectuée dans l'environnement LSIS_DME. Nous avons créé plusieurs jeux d'événements planifiés en entrée et plusieurs paramétrages des modèles atomiques de traitement des tâches pour tester les limites de la configuration des procédures H80SH27F. Pour des raisons de confidentialité, ces jeux de données et les paramétrages des modèles ne proviennent pas d'un cas réel. Les événements d'entrée (représentant le produit FDHB2AAG-03) ont été initialisés avec des valeurs permettant de définir certaines caractéristiques. Par exemple, un pourcentage donné de FDHB2AAG-03 a été défini comme défectueux en sortie de certaines opérations.

Nous présentons dans le tableau suivant quelques résultats de simulation pour des jeux de cent items FDHB2AAG-03 en entrée du modèle H80SH27F.

	Items défectueux en Oper 1020	Niveau Stock Moyen en Oper 1020	Durée moyenne de transit des Items en unités de temps
Jeu 1	1	<1	≈50
Jeu 2	10	≈3	≈54
Jeu 3	20	≈6	≈58
Jeu 4	30	≈10	≈65

Figure 104 - Résultat G-DEVS couplé de la route H80SH27F sur LSIS_DME

Dans ce Workflow simple, nous observons, en particulier, la durée moyenne de transit des items. Ces données sont déterminées à partir de la trace de simulation contenant les événements reçus, traités et générés, du modèle couplé G-DEVS de la route H80SH27F.

Ces informations permettent de déterminer une planification judicieuse des items FDHB2AAG-03 et peuvent indiquer une remise en question des enchaînements de tâches. En particulier, l'analyse du Stock de l'Oper 1020 nous indique un niveau-moyen élevé, dans le cas d'un nombre d'item défectueux égal à trente pourcents. A partir de ce constat dans ce cas simple et dans l'hypothèse de trente pourcents de produits défectueux, nous proposons de paralléliser l'Oper 1020. Le nouveau modèle indique, par simulation, que l'on peut réduire ainsi la durée de transit à environ 56 unités de temps.

5.4 Définition du Workflow compatible HLA

Les conditions d'utilisation de l'environnement Workflow peuvent poser les contraintes suivantes :

- tout d'abord, les applications qui sont interfacées avec le moteur de l'environnement peuvent être hétérogènes.
- De plus, les actions de monitoring et d'administration menées sur certaines informations du modèle Workflow en cours d'exécution peuvent être exécutées sur un ordinateur distant.

Nous avons présenté précédemment un cadre conceptuel en réponse à ce type de pré requis. Ce cadre repose sur la compatibilité à la norme HLA des modèles G-DEVS qui permet d'interfacer ces modèles avec les différentes applications de monitoring, administration, clientes et invoquées, elles même rendues compatibles HLA. Pour cela, il est nécessaire de définir un certain nombre d'objets partagés dans la fédération représentant l'environnement Workflow. Le comportement global sera donc régi par un mécanisme d'échange de messages entre des processeurs distants. Nous présentons sur la Figure 105 [ZACHAREWICZ 06b], les différents outils d'un Workflow s'interfaçant avec HLA dans le cas concret de la modélisation de la route H80SH27F.

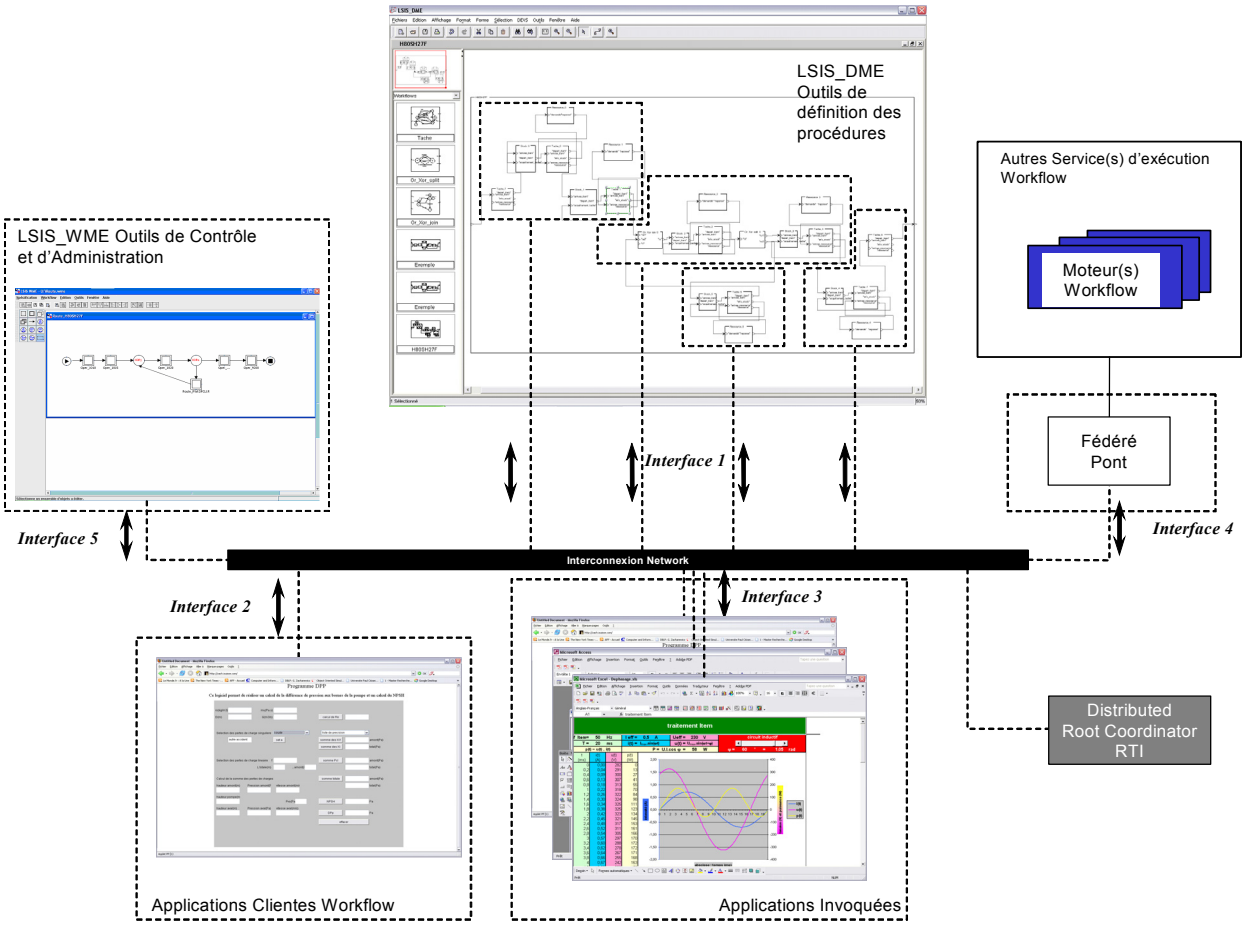


Figure 105 - Modèle référence de L'environnement Workflow avec les interfaces

5.4.1 Création des tables d'un FOM HLA

5.4.1.1 Création des tables d'objet HLA

Les classes spécifiées pour la fédération Workflow doivent être spécialisées en fonction de l'exemple considéré, à savoir dans notre cas la route H80SH27F. L'interface 1 possède ici deux classes filles permettant de récupérer les informations de l'état actuel du modèle et des Items qui circulent sous forme d'événement dans le modèle G-DEVS pendant la simulation.

L'interface 2 possède deux classes permettant de récupérer les informations issues d'une interface homme-machine, précisément pour définir le routage du Xoj d'une part en incluant

des informations dans l'Item avant de l'injecter dans le composant de choix, et d'autre part en modifiant l'Item manuellement dans une opération non automatique.

L'interface 3 récupère les informations générées par les applications impliquées automatiquement dans le Workflow, précisément le traçage d'une courbe de productivité et le stockage de données dans une base de données.

L'interface 4 peut permettre de diffuser ou de récupérer des informations provenant d'une autre fédération Workflow.

Enfin l'interface 5 permet de prendre en compte les modifications sur le processus (en terme de structure et de paramétrage) qui peuvent être apportées par l'outil de monitoring et d'administration.

Object Class Structure Table				
HLA object Root (N)	Interface (S)	Interface 1 (PS)	Contenu Evenements (PS)	Contenu Item (PS)
				Item de Sortie (PS)
			Table d'etat actuel des modèles (PS)	Contenu des Stocks (PS)
			Information Debut / Fin de simulation (PS)	Indication de début/fin de Oper 1015 (PS)
				Indication de début/fin de Oper … (PS)
				Indication de début/fin de Route_RWKDFCL1R (PS)
		Interface 2 (S)	Modification du contenu d'Item pour routage (PS)	Modification du contenu d'Item pour routage Xoj apres 1015 (PS)
			Modification du contenu d'Item par operation non gérée par la Workflow (PS)	Modification du contenu d'Item par operation non gérée par la Workflow (PS)
		Interface 3 (S)	Information provenant de l'aplication invoquée (PS)	Traitment automatique d'un partie de la Route RWKDFLC1R (PS)
		Interface 4 (S)	Information vers le fédéré pont appartennant aux deux Federation de Workflow (PS)	non défini
		Interface 5 (S)	Information d'administration en modification du modèle (PS)	Modification Oper 1010 (PS)
				Modification Oper 1015 (PS)
				Modification Oper … (PS)
				Modification Oper 9020 (PS)
				Modification Oper 1020 (PS)
				Modification Route RWKDFCL1R (PS)
				Modification couplage H80SH27F (PS)
			Information d'administration en parametrage des application invoquées (PS)	Réglage de paramètre sous Excel pour traitement la Route RWKDFCL1R (PS)

Figure 106 - Table d'objet HLA de la fédération environnement Workflow

5.4.1.2 Création des tables d'attributs HLA

Les attributs des différents objets partagés dans la fédération sont présentés ci dessous dans la Figure 107.

Attribute Table									
Object	**Attribute**	**Data-type**	**Update Type**	**Update Condition**	**D/A**	**P/S**	**Available Dimensions**	**Transportation**	**Order**
HLAobjectRoot	HLA privilege toDelete Object	NA	NA	NA	N	N	NA	HLAreliable	Time Stamp
Contenu Item	Date	TimeType	Conditional	Workflow	N	PS	NA	HLAreliable	Time Stamp
	Destinataire	HLAASCII string	Conditional	Workflow	N	PS	NA	HLAreliable	Time Stamp
	Contenu	HLAvariableArray	Conditional	Workflow	N	PS	Polynimial Dimension	HLAreliable	Time Stamp
Item de Sortie (PS)	Date	TimeType	Conditional	Workflow	N	PS	NA	HLAreliable	Time Stamp
	Destinataire	HLAASCII string	Conditional	Workflow	N	PS	NA	HLAreliable	Time Stamp
	Contenu	HLAvariableArray	Conditional	Workflow	N	PS	Polynimial Dimension	HLAreliable	Time Stamp
Contenu des Stocks (PS)	Dimansion Liste	HLAInteger16BE	static	NA	N	PS	NA	HLAreliable	Time Stamp
	Liste d'Items	HLAvariableArray	Conditional	Workflow	N	PS	List Dimension	HLAreliable	Time Stamp
Indication de début/fin de Oper 1015 (PS)	Date de debut	TimeType	Conditional	Workflow	N	PS	NA	HLAreliable	Time Stamp
	Date de fin	TimeType	Conditional	Workflow	N	PS	NA	HLAreliable	Time Stamp
Indication de début/fin de Oper … (PS)	Date de debut	TimeType	Conditional	Workflow	N	PS	NA	HLAreliable	Time Stamp
	Date de debut	TimeType	Conditional	Workflow	N	PS	NA	HLAreliable	Time Stamp
Indication de début/fin de Route_RWKDFCL1R (PS)	Date de debut	TimeType	Conditional	Workflow	N	PS	NA	HLAreliable	Time Stamp
	Date de debut	TimeType	Conditional	Workflow	N	PS	NA	HLAreliable	Time Stamp
Modification du contenu d'Item pour routage Xoj apres 1015 (PS)	Date	TimeType	Conditional	Workflow	N	PS	NA	HLAreliable	Time Stamp
	Destinataire	HLAASCII string	Conditional	Workflow	N	PS	NA	HLAreliable	Time Stamp
	Contenu	HLAvariableArray	Conditional	Workflow	N	PS	Polynimial Dimension	HLAreliable	Time Stamp
Modification du contenu d'Item par operation non gérée par la Workflow (PS)	Date	TimeType	Conditional	Workflow	N	PS	NA	HLAreliable	Time Stamp
	Destinataire	HLAASCII string	Conditional	Workflow	N	PS	NA	HLAreliable	Time Stamp
	Contenu	HLAvariableArray	Conditional	Workflow	N	PS	Polynimial Dimension	HLAreliable	Time Stamp
Traitment automatique d'un partie de la Route RWKDFLC1R (PS)	Liste d'Items générés par l'application invoquées	HLAvariableArray	Conditional	Workflow	N	PS	List Dimension	HLAreliable	Time Stamp
Modification Oper 1010 (PS) (idem pour les Oper 1015, 1020, …, 9200)	Listes des phases	HLAvariableArray	Conditional	Workflow	N	PS	List Dimension	HLAreliable	Time Stamp
	Listes des transitions internes	HLAvariableArray	Conditional	Workflow	N	PS	List Dimension	HLAreliable	Time Stamp
	Listes des transitions externes	HLAvariableArray	Conditional	Workflow	N	PS	List Dimension	HLAreliable	Time Stamp
	Listes des fonctions de sortie	HLAvariableArray	Conditional	Workflow	N	PS	List Dimension	HLAreliable	Time Stamp
	Listes des durées de vie	HLAvariableArray	Conditional	Workflow	N	PS	List Dimension	HLAreliable	Time Stamp
Modification Route RWKDFCL1R (PS)	Listes des Modèle inclus	HLAvariableArray	Conditional	Workflow	N	PS	List Dimension	HLAreliable	Time Stamp
	Listes des EIC	HLAvariableArray	Conditional	Workflow	N	PS	List Dimension	HLAreliable	Time Stamp
	Listes des EOC	HLAvariableArray	Conditional	Workflow	N	PS	List Dimension	HLAreliable	Time Stamp
	Listes des IC	HLAvariableArray	Conditional	Workflow	N	PS	List Dimension	HLAreliable	Time Stamp
	Select	HLAvariableArray	Conditional	Workflow	N	PS	Model Dimension	HLAreliable	Time Stamp
Modification couplage H80SH27F (PS)	Listes des EIC	HLAvariableArray	Conditional	Workflow	N	PS	List Dimension	HLAreliable	Time Stamp
	Listes des EOC	HLAvariableArray	Conditional	Workflow	N	PS	List Dimension	HLAreliable	Time Stamp
	Listes des IC	HLAvariableArray	Conditional	Workflow	N	PS	List Dimension	HLAreliable	Time Stamp
Réglage de paramètre sous Excel pour traitement la Route RWKDFCL1R (PS)	Liste de paramètres	HLAvariableArray	Conditional	Workflow	N	PS	Polynimial Dimension	HLAreliable	Time Stamp

Figure 107 - Tables d'attributs HLA de L'environnement Workflow

Ces attributs contiennent les informations concernant les items échangés, les informations échangées avec les applications invoquées et les applications clientes et enfin les informations concernant les modifications de structure du Workflow.

5.4.2 Mécanisme de Publication/ Souscription

5.4.2.1 Application de Monitoring et d'administration

Un environnement Workflow doit comporter un outil de monitoring et d'administration. Nous avons choisi d'intégrer l'outil de modélisation LSIS_WME. En effet, cet outil de représentation graphique peut être étendu pour afficher une animation, permettant ainsi de suivre l'évolution de la simulation. Cet outil peut également choisir d'arrêter la simulation, de modifier la structure du modèle de la procédure Workflow et de relancer la simulation à partir d'un point (un état) donné en tenant compte des modifications de structure.

La norme HLA est couramment employée pour intégrer les fonctions de monitoring sur des applications distantes et/ou hétérogènes. De nombreux exemples sont présentés dans la littérature pour illustrer ce type d'application ; nous retiendrons l'application de modélisation et simulation de Flotte navale du groupe de modélisation Italien SIREN [REVETRIA 03].

Le fédéré de monitoring et administration incluant l'application LSIS_WME souscrira donc à des informations fournies par les fédérés assurant la simulation. De son côté, il assurera la diffusion de modifications de la structure du Workflow en publiant un objet décrivant la nouvelle structure du modèle.

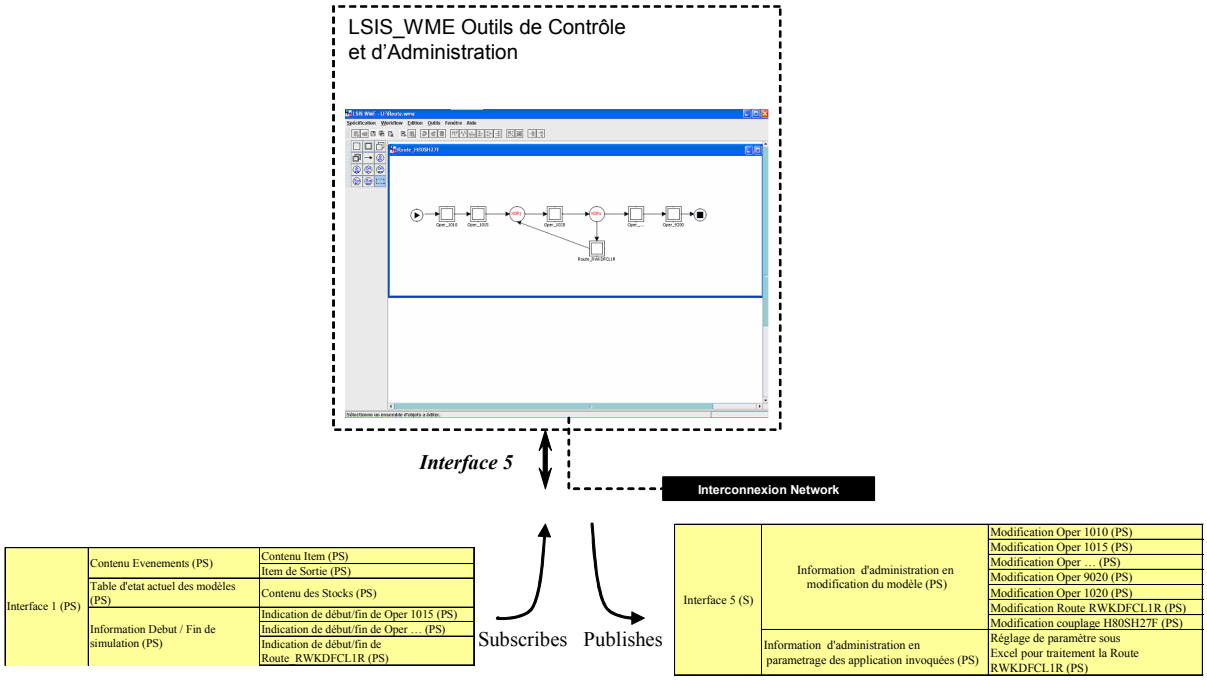


Figure 108 - Maquette des interfaces graphiques de monitoring

5.4.2.2 Applications invoquées et applications clientes

Un environnement Workflow doit être interfacé avec des applications et des outils qui seront appelés pour effectuer un travail, en cours de simulation, sur les items en circulation dans le Workflow. Un utilisateur humain peut également entrer des données sur une interface graphique, à un point donné de la simulation, pour effectuer des choix de chemins par exemple, ou modifier des données de façon « ad hoc » ou encore effectuer un choix qualitatif sur les

données traitées non prises en compte par l'environnement. Enfin des applications effectuant des traitements automatiques sur les données peuvent être invoquées. Dans notre cas le traçage d'une courbe de productivité et le stockage de données dans une base de données sera effectué par des applications externes.

Les fédérés des applications souscriront donc aux informations fournies par les fédérés assurant la simulation. De leur coté, ils assureront la diffusion des informations provenant d'un utilisateur humain ou de programmes appelés en publiant un objet décrivant les nouvelles informations à prendre en compte dans la suite de la simulation du modèle.

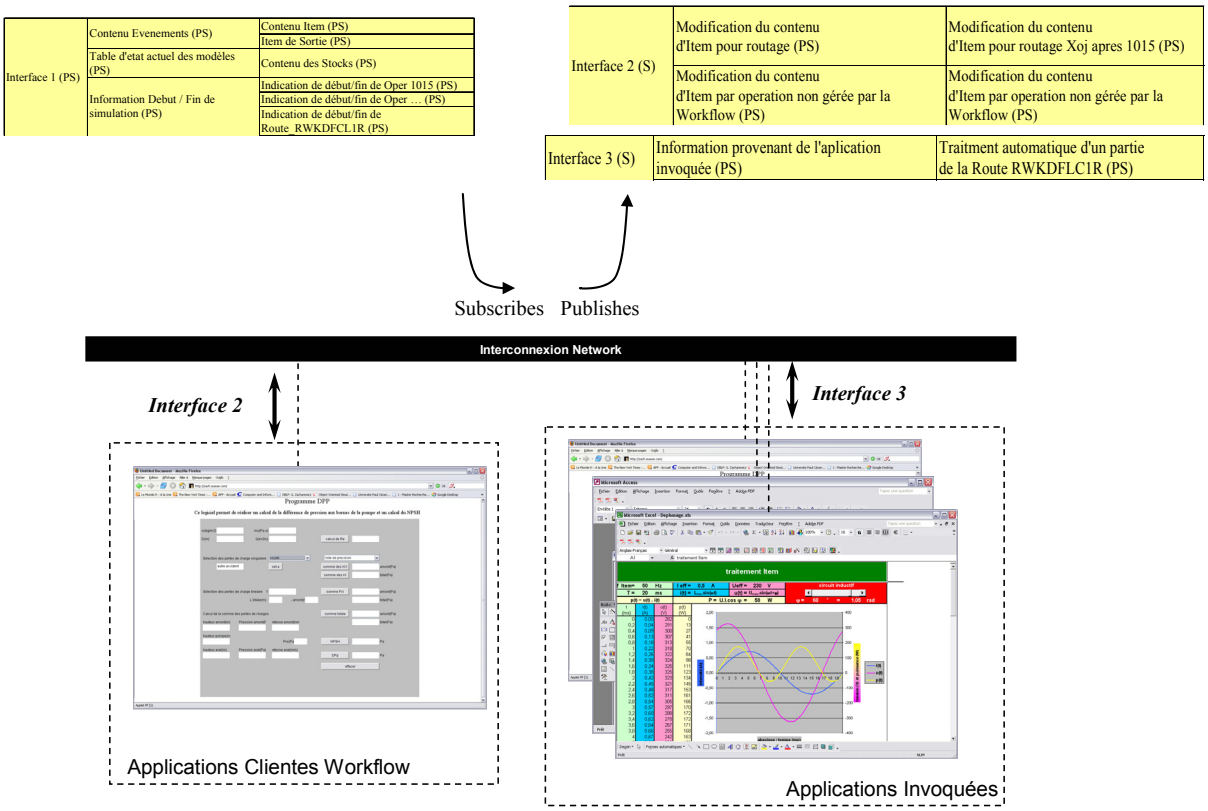


Figure 109 - Maquette des interfaces graphiques des applications

5.4.2.3 Communication avec d'autres Workflow

Nous avons également pris en considération la possibilité d'interfaçage avec un autre environnement Workflow par l'intermédiaire d'un fédéré pont à partir des travaux de communications entre fédérations proposés par [BRÉHOLÉE 03].

5.5 Conclusion de la modélisation de la route H80SH27F

L'illustration de l'environnement Workflow G-DEVS / HLA sur un cas concret issu du monde industriel en formalisme G-DEVS dans un contexte distribué HLA, nous a permis de valider la cohérence de la structure du modèle de référence Workflow présenté Figure 105. En effet, les résultats de simulation nous ont fourni des informations cohérentes notamment sur la durée de transit des items, les niveaux des stocks en cours de simulation, les goulots d'étranglement pouvant se produire sur une chaîne, etc.

Notons que dans le cas de la modélisation des Workflow nous utilisons l'algorithme §3.3.3 du chapitre 2. Cependant, nous pouvons retenir que, dans le cas de la définition d'un environnement Workflow, HLA est plus sollicité sur le plan de l'interopérabilité que de la performance. Malgré cela, les modèles Workflow de base appartiennent aux classes GDEVS-$_{PF1}$ et GDEVS$_{PF2}$, sur lesquelles les algorithmes §3.4.1 et §3.4.2 du chapitre 2 peuvent être appliqués.

Ces résultats seront employés pour assister le processus décisionnel humain impliqué dans le cadre de la modification des flux de procédures dans les chaînes de production STMicroelectronics. Ils permettront d'anticiper les erreurs qui peuvent se produire lors d'une mauvaise modification d'un flux de procédures. Il permettra également de comparer quantitativement plusieurs modifications d'une procédure pour déterminer la plus efficace.

6 Perspectives

Dans ce chapitre, nous avons défini les parties de modélisation, simulation et supervision essentielles à la définition d'un environnement Workflow. Cependant le développement et l'intégration de modules supplémentaires dans l'environnement Workflow peut être envisagé. Cette intégration devra être assurée par une insertion de ces modules dans des fédérés HLA. La perspective de couplage avec d'autres Workflow de l'environnement est également proposée dans ce chapitre. Dans chacun de ces couplages, les données à échanger entre les fédérations Workflow devront être précisées.

De plus, les Workflow sont construits à partir de recueils d'information provenant d'experts du domaine. Ces derniers évoquent plus couramment les données temporelles en termes de plages de valeurs (ou de valeurs moyennes) plutôt que de valeurs exactes. Il serait donc intéressant de considérer la modélisation par les Min-Max DEVS [GIAMBIASI 01] et [HAMRI 05].

Enfin, le couplage de l'environnement avec des procédures réelles est également une perspective. Cette solution nécessite alors, la mise en œuvre de capteurs d'informations provenant du système réel. Ces données devront être intégrées à la simulation temps réel qui aura pour contrainte d'être au moins aussi rapide dans l'exécution du temps simulé que le temps d'horloge murale [FUJIMOTO 00].

7 Conclusion

Les entreprises de dimensions de plus en plus importantes, en demande de flexibilité, de contrôle et dont la répartition devient planétaire, nécessitent des applications en accord avec ces réalités. Pour répondre à ces attentes un champ d'applications récent est apparu : les Workflow. La croissance très rapide, cette décennie, du nombre de logiciels applicatifs développés, décline que le domaine s'avère très porteur pour les secteurs tant industriels qu'administratifs.

Nous avons montré que HLA offre un complément de réponse pour gérer les composants d'un système Workflow distribué, ce qui n'est pas traité par la spécification Workflow originale. En outre, la faible bibliographie au croisement des Workflow et d'HLA atteste des pers-

pectives possibles. Enfin, l'utilisation du formalisme G-DEVS pour représenter un Workflow apporte la clarté et la non-ambigüité d'une spécification formelle.

Sur le plan de la réalisation, le développement en Java, de l'environnement de modélisation, simulation et supervision ainsi que du RTI rend ces programmes portables sur les systèmes d'exploitation les plus courants (Windows, Linux, Mac OS, …).

Enfin, la programmation par composants compatibles HLA, permet de définir des bibliothèques de fédérés modulaires. Ces fédérés pourront être intégrés à d'autres fédérations. De plus, les propriétés d'interopérabilité fournies par HLA nous ont permis de réutiliser les outils de modélisation de Workflow LSIS_WME et LSIS_DME.

Conclusion générale

Nous avons défini dans ce mémoire un environnement de modélisation & simulation basé sur les formalismes DEVS et G-DEVS, pourvu d'une extension pour la définition de modèles distribués compatibles HLA. Les travaux présentés concernent principalement trois points :

- La proposition d'améliorations d'algorithmes de simulation distribuées conservatives de modèles G-DEVS (Generalised Discrete Event Specification) [GIAMBIASI 00] et DEVS [ZEIGLER 76],
- La définition et la réalisation d'un environnement de modélisation & simulation de modèles G-DEVS compatible HLA (High Level Architecture) implémentant les améliorations proposées,
- L'application de l'environnement à la modélisation et la simulation de Workflow.

Sur le premier point, nous avons proposé un simulateur G-DEVS conceptuel dont les fonctions de communication et de synchronisation entre les composants distribués respectent la norme HLA. Nous avons introduit un coordinateur distribué incluant un algorithme de communication avec le RTI HLA, ce qui permet, entre autres, de conserver la modularité et l'indépendance des modèles par rapport à la méthode de simulation. Cette première amélioration significative, basée sur le mécanisme de synchronisation conservative, permet, en outre, d'utiliser un Lookahead positif. Nous avons ensuite proposé deux algorithmes originaux pour le calcul d'un Lookahead relatif à l'état courant d'un modèle distribué. Ces algorithmes permettent d'augmenter très notablement les performances de la simulation distribuée comme l'illustrent les expériences et les développements que nous avons menés. Dans ces approches, le premier algorithme a été appliqué à la classe des modèles $G\text{-}DEVS_{PF1}$ qui possèdent une fonction de durée de vie des états dépendant d'une seule variable d'état et le second algorithme a été appliqué aux modèles $G\text{-}DEVS_{PF2}$ qui possèdent plusieurs variables d'état. Ces algorithmes sont basés sur l'analyse du domaine de variation de la fonction de durée de vie impliquée dans la détermination du Lookahead du modèle considéré.

Pour le deuxième point, nous avons détaillé, le développement de l'environnement de modélisation et la simulation distribuée de modèles G-DEVS compatibles HLA basé sur les approches introduites dans le second chapitre de cette thèse. Des mesures de performance ont été effectuées sur cet environnement afin de quantifier d'une part l'efficacité de la mise à plat de la structure de simulation et d'autre part, l'accélération engendrée par l'utilisation d'un Lookahead positif. Les simulations distribuées dirigées par les événements, fondées sur un mécanisme de synchronisation conservatif et utilisant nos algorithmes de communications et de calcul de Lookahead, se sont avérées performantes, elles permettent une accélération de l'ordre de quarante pourcents pour les modèles mis en œuvre.

L'environnement a été intégré à une application de Workflow à laquelle nous avons adjoint un module de transformation automatique de Workflow en modèles G-DEVS. Ce nouvel environnement apporte au domaine des Workflow, l'aspect formel de G-DEVS et la flexibilité de la simulation distribuée HLA. Des applications de cet environnement dédiées à des cas réels ont validé les possibilités offertes. Les performances de l'approche proposée ouvrent le champ de cette contribution au monde industriel.

Dans le prolongement de ces travaux il nous paraîtrait intéressant de pouvoir étendre la classe de modèle G-DEVS compatibles HLA sur laquelle peut porter le calcul de Lookahead. Ce calcul étant jusqu'alors défini pour les classes $G\text{-}DEVS_{PF1}$ et $G\text{-}DEVS_{PF2}$. Pour cela, pourraient être envisagés d'autres types de fonctions de durée de vie d'état D(s), par exemple des fonctions contraintes dépendantes de plusieurs variables réelles estimées et sur lesquelles seraient appliquées des théorèmes mathématiques pour l'étude de leurs domaines de variation.

Par ailleurs, au niveau de l'environnement LSIS_DME, certains utilisateurs ont émis le souhait de pouvoir définir des modèles, dont les caractéristiques sont modélisables sous la forme d'algorithmes de programmation, en particulier pour la définition des fonctions de transition et de certains modèles à états implicites. Pour répondre à ce besoin, des travaux consistant à définir un langage textuel de description des modèles, indépendamment de l'éditeur de l'environnement LSIS_DME viennent de débuter et s'inscrivent dès lors dans le prolongement de cette thèse.

Enfin, nous avons détaillé dans le dernier chapitre de ce mémoire les parties essentielles de modélisation, simulation et supervision d'un environnement Workflow. Il pourrait être également souhaitable de compléter cet environnement par l'ajout de modules spécifiques permettant d'intégrer des applications invoquées ou des interfaces homme-machine. Cette intégration serait facilitée par la compatibilité HLA de l'environnement. Pour conclure, les Workflow sont construits à partir de recueils d'information provenant d'experts des domaines, qui évoquent plus fréquemment les tâches d'un processus en termes d'intervalles temporels que de durées exactes. Pour cette raison, il nous apparaîtrait très intéressant d'envisager au plan contributif d'autres développements par une modélisation des procédures par le formalisme Min-Max DEVS [GIAMBIASI 01a] et [HAMRI 05] notamment.

Bibliographie

[ADER 1996] Ader, M. *Management collectif de l'information*, 1996.

[BORLAND 02] Borland. JBuilder9, Borland Software Corporation, 100 Enterprise Way, Scotts Valley, CA 95066-3249, 2002, http://www.borland.com.

[BRÉHOLÉE 03] Bréholée, B. and Siron, P., Bridges in HLA distributed simulations. in *2003 International Industrial Simulation Conference*, (Valencia, Spain, 2003).

[BRYANT 77] Bryant, R.E. Simulation of packet communication architecture computer systems, MIT, 1977.

[BUBAK 03] Bubak, M., Gubała, T., Malawski, M. and Rycerz, K., Grid Application Workflow Composition. in *The Nineth Global Grid Forum, "GGF@Work"*, (2003).

[CALVIN 96] Calvin, J.O. and Weatherly, R., An introduction to the high level architecture (HLA) runtime infrastructure (RTI). in *4th DIS Workshop on Standards for the Interoperability of Defense Simulations*, (Orlando, FL: University of Central Florida, 1996), 705-715.

[CARSTEN 94] Carsten, T., Interface Oriented Classification of DEVS Models. in *AI, Simulation, and Planning in High Autonomy Systems, 1994. 'Distributed Interactive Simulation Environments'*, (Gainesville, FL, USA, 1994), 208-213.

[CHANDY 79] Chandy, K.M. and Misra, J. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, *5* (5). 440-452, 1979.

[CHANDY 81] Chandy, K.M. and Misra, J. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, *24* (4). 198-206, 1981.

[CHIKINE 93] Chikine, E., Kissélev, A., Krasnov, M. and Makarenko, G. *Mathématiques supérieures- pour ingénieurs et polytechniciens*, 1993.

[CHOW 94] Chow, A.C.H. and Zeigler, B.P., Parallel DEVS: a parallel, hierarchical, modular, modeling formalism. in *26th conference on Winter simulation*, (Orlando, Florida, United State, 1994), Society for Computer Simulation International, 716 - 722.

[CLOG 06] C-Log. Workey, C-Log International 480, Place de la Courtine 93194 NOISY LE GRAND, 2006, http://www.c-log.com/040_Workey.

[CORMEN 02] Cormen, T., Leiserson, C. and Rivest, R. *Introduction à l'algorithmique*. Dunod, 2002.

[COSA 02] Cosa. COSA BPM, 50259 Pulheim Germany, 2002, http://www.cosa-bpm.com.

[COURTOIS 96] Courtois, T. Workflow: la gestion globale des processus de l'entreprise. *Logiciels & Systèmes*, *11*, 1996.

[DMSO 98a] Defense Modeling And Simulation Office. High Level Architecture Rules, 2002.

[DMSO 98b] Defense Modeling And Simulation Office. High Level Architecture Interface Specification, 2002.

[DMSO 98c] Defense Modeling And Simulation Office. Object Model Template Specification, 1998.

[DMSO 98d] Defense Modeling And Simulation Office. High Level Architecture Federation Development and Execution Process (FEDEP) Model, 1998.

[DMSOe 98e] Defense Modeling And Simulation Office. Object Model Data Tool (OMDT), 1998.

[DMSO 05] Defense Modeling And Simulation Office. RTI Verification Status Board Technology Transition: High Level Architecture, 2005, https://www.dmso.mil/public/transition/hla/statusboard.

[DESEL 00] Desel, J., Teaching system modeling, simulation and validation. in *2000 Winter Simulation Conference*, (2000), J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick.

[DIJKSTRA 59] Dijkstra, E.W. A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*. 269-271, 1959.

[DIS 94] The DIS Steering Committee. The DIS vision: A map to the future of distributed simulation, University of Central Florida, Orlando, 1994.

[DROST 01] Drost, J. ShiftOne JRat (Runtime Analysis Toolkit), Logiciel GNU, 2001, http://jrat.sourceforge.net/.

[ECLIPSE 04] Eclipse. Eclipse, Eclipse Foundation, Inc. 102 Centrepointe Drive Ottawa, Ontario, Canada, K2G 6B1, 2004, http://www.eclipse.org/.

[EDER 98] Eder, J. and Liebhart, W., Contributions to Exception Handling in Workflow Management. in *EDBT Workshop on Workflow Management Systems*, (Valencia, 1998), 3 - 10.

[ESCUDE 00] Escude, B. Modélisation et Simulation à événements discrets de Systèmes Hybrides *DIAM*, Université de Droit, d'Economie et des Sciences d'Aix-Marseille III, Marseille, 2000.

[FILIPPI 04] Filippi, J. and Bisgambiglia, P. JDEVS: an implementation of a DEVS based formal framework for environmental modelling. *Environmental Modelling & Software, Elsevier Science*, *19* (3). 261-274, 2004.

[FLOYD 62] Floyd, R.W. Algorithm 97: Shortest path. *Communication of the ACM*, *5* (6). 345, 1962

[FUJIMOTO 97] Fujimoto, R.M., Zero Lookahead and repeatability in the high level architecture. in *Spring Simulation Interoperability Workshop (SIW)* (Orlando, FL, 1997).

[FUJIMOTO 98] Fujimoto, R.M. Time management in the high level architecture. *Simulation*, *71* (6). 388 - 400, 1998.

[FUJIMOTO 00] Fujimoto, R.M. *Parallel discrete event simulation*. Wiley Interscience, New York, NY, 2000.

[GEORGAKOPOULOS 95] Georgakopoulos, D., Hornick, M.F. and A.P.Sheth An overview of workflow management from process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, *3* (2). 119 - 153, 1995.

[GIAMBIASI 79] Giambiasi, N., Miara, A. and Muriach, D., SILOG: A Practical Tool for Large Digital Network Simulation. in *16th Annual ACM IEEE Conference on Design Automation*, (San Diego, USA, 1979), IEEE Press, 263-271.

[GIAMBIASI 95] Giambiasi, N., Frydman, C. and Escudé, B., Hierarchical/Multi-View Modelling and Simulation. in *7th European Simulation Symposium* (Erlangen Nuremberg - Germany., 1995).

[GIAMBIASI 00] Giambiasi, N., Escude, B. and Ghosh, S. G-DEVS A Generalized Discrete Event Specification for Accurate Modeling of Dynamic Systems. *Transactions of the Society for Computer Simulation International*, *17* (3). 120 - 134, 2000.

[GIAMBIASI 01a] Giambiasi, N. and Ghosh, S., Min-Max DEVS: A new formalism for the specification of discrete event models with Min-Max delays in *Conference Proceedings European Simulation Symposium*, (Marseille, France, 2001), 616-621.

[GIAMBIASI 01b] Giambiasi, N. Cours DEA MCAO Université Aix Marseille III, 2001.

[GIRE 06] Gire, S. Cours d'Algorithmique et Structure de données, 2006, http://fastnet.univ-brest.fr/~gire/COURS/ALGO_C/node2.html.

[GLINSKY 02a] Glinsky, E. and Wainer. G., Performance Analysis of Real-Time DEVS models. in *Proceedings of 2002 Winter Simulation Conferenc*e. (San Diego, CA. USA. 2002).

[GLINSKY 02b] Glinsky, E. and Wainer. G., Performance analysis of DEVS environments. In *Proceedings of AIS'2002*. (Lisbon, Portugal. 2002).

[GLINSKY 05] Glinsky, E. and Wainer, G.A., DEVStone: a Benchmarking Technique for Studying Performance of DEVS Modeling and Simulation Environments. in *9-th IEEE International Symposium on Distributed Simulation and Real Time Applications*, (Montreal, Canada, 2005).

[GLINSKY 06] Glinsky, E. and Wainer. G., New Parallel simulation techniques of DEVS and Cell-DEVS in CD++. in *Proceedings of the 38th IEEE/SCS Annual Simulation Symposium*. (Huntsville, AL, 2006).

[HAMRI 05] Hamri, M., Giambiasi, N. and Frydman, C. Simulation semantics for Min-Max DEVS models. *Lecture Notes in Computer Science, LNAI 3397*. 699 - 708, 2005.

[HAN 98] Han, Y., Sheth, A. and Bussler, C., A Taxonomy of Adaptive Workflow Management. in *ACM CSCW 98 workshop, Towards Adaptive Workflow Systems*, (Seattle, W, 1998), Christoph Bussler.

[HAYES 91] Hayes, K. and Lavery, K. Workflow management software: the business opportunity., Ovum Ltd, London, 1991.

[IEEE1516 00] IEEE std 1516-2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules. Engineers, T.I.o.E.a.E. ed., New York, NY, 2001.

[IEEE1516.1 00] IEEE std 1516.1-2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification. Engineers, T.I.o.E.a.E. ed., New York, NY, 2001.

[IEEE1516.2 00] IEEE std 1516.2-2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification. Engineers, T.I.o.E.a.E. ed., New York, NY, 2001.

[IEEE1516.3 03] IEEE std 1516.3-2003. IEEE Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP). Engineers, T.I.o.E.a.E. ed., New York, NY, 2003.

[INCONCERT 97] InConcert. InConcert Specifications Document, 3303 Hillview Avenue Palo Alto, CA 94304 USA, 1997, http://www.inconcert.com.

[JEFFERSON 85] Jefferson, D.R. Virtual Time. *ACM Transaction on Progamming Langage and Systems*, *7* (3). 404 - 425, 1985.

[JENSE 97] Jense, G.J., Kuijpers, N.H.L. and Dumay, A.C.M. DIS and HLA: connecting people, simulations and simulators in the military, space and civil domains. Astronautics., TNO Physics and Electronics Laboratory, 1997.

[JGRAPH 06] JGraph, L. JGraph, JGraph Ltd., 73 Bridgewater Drive Northampton England, NN3 3AF, 2006, http://www.jgraph.com.

[JOOSTEN 96] Joosten, S. and Brinkkemper, S., Fundamental Concepts for Workflow Automation in Practice. in *5th International Conference Information Systems Development*, (Amsterdam, 1996).

[KIM 00] Kim, K., Kang, W., Sagong, B. and Seo, H., Efficient Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-Hierarchical One. in *33rd Annual Simulation Symposium*, (Washington, D.C., 2000), 227.

[KLIR 85] Klir, G.J. *Architecture of Systems Problem Solving*. Plenum Press, New York, 1985.

[KOULOPOULOS 95] Koulopoulos, T.M. *The Workflow Imperative*. Van Nostrand Reinhold, New York, 1995.

[LAKE 00] Lake, T., Zeigler, B.P., Sarjoughian, H.S. and Nutaro, J., DEVS Simulation and HLA Lookahead. in *Simulation Interoperability Workshop (SIW)*, (Orlando, FL, 2000).

[LAMPORT 78] Lamport, L. Time, clocks and the ordering of events in a distributed system. *Communication of the ACM*, *21* (7). 558-565, 1978.

[LSIS 05] LSIS. LSIS_DME, LSIS, Marseille, France, 2005, http://www.site.uottawa.ca/~oren/SCS_MSNet/coop-2005.htm.

[MCLEOD 99] McLeod Institute of Simulation Sciences. Cours en ligne, Université Chico, 1999, http://www.ecst.csuchico.edu/~hla/courses.html.

[McCREADY 92] McReady, S. There is more than one kind of Workflow software. *Computerworld*, *2* 86 - 90, 1992.

[MIARA 78] Miara, A. and Giambiasi, N., Dynamic and deductive fault simulation. in *In Proceedings of the 15th Conference on Design Automation Conference*, (Las Vegas, Nevada, USA, 1978), IEEE Press, 439 - 443.

[MILLER 95] Miller, G., Adams, A. and Fischer, M.C. Aggregate Level Simulation Protocol (ALSP) Project, The MITRE Corporation, 1995.

[MINSKY 56] Minsky, M. *Some Universal Elements For Finite Automata in Automata Studies*. Princeton University Press, 1956.

[NUTARO 03] Nutaro J. J., *Parallel Discrete Event Simulation with Application to Continuous System*s, Ph. D. Dissertation, Fall 2003, Electrical and Computer Engineering Dept, Univerisity of Arizona

[OREN 87] TI Orën, *Taxonomy of simulation Model processing* in encyclopedia of systems and control (eds. M. Singh), Pergamon Press, 1987

[PEREZ 05] Perez, C.E. Open Source Workflow Engines Written in Java, 2005, http://www.manageability.org/blog/stuff/workflow_in_java.

[PETRI 62] Petri, C.A. Kommunikation mit Automaten *Institut für instrumentelle Mathematik*, Schriften des IIM, Bonn, 1962.

[PITCH 02] Pitch. pRTI 1516, Pitch Technologies AB Nygatan 35 S-582 19 Linköping Sweden, 2002, http://www.pitch.se/prti1516/default.asp.

[REVETRIA 03] Revetria, R., NAVI: Learning HLA from an Interactive Exercise Tutorial. in *SCSC 2003* (Montreal, Canada, 2003).

[SADIQ 99] Sadiq, S.W. Workflows in Dynamic Environments Can they be manage? *Computer science and Electrical Engineering*. 165 - 176, 1999.

[SAMADI 85] Samadi, B. Distributed simulation, algorithms and performance analysis, UCLA, Los Angeles, CA, USA, 1985.

[SCHAEL 97] Schaël, T. *Théorie et Pratique du Workflow*. Springer-Verlag, 1997.

[SCHEER 97] Scheer, A.W., Borowsky, S., Klabunde, S. and Traut., A., Flexible industrial applications through model-based Workflows. in *the International Conference on Enterprise Integration and Modeling Technology*, (Torino, Italy, 1997), Springer, 439 - 448.

[SECK 05] Seck, M., Frydman, C. and Giambiasi, N. Using DEVS for Modeling and Simulation of Human Behavior. *Lecture Notes in Computer Science special issue DEVS modelling and simulation*, *3397*. 692-698, 2005.

[SONG 94] Song, H.S. and Kim, T.G., The DEVS framework for discrete event systems control. in *5th Annual Conference on AI, Simulation and Planning in High Autonomous Systems*, (Gainesville, FL, USA, 1994), 228 - 234.

[STAFFWARE 01] Staffware. Staffware, Palo Alto, California, 2001, http://www.staffware.com, http://www.tibco/com.

[TARDIEU 85] Tardieu, H., Colletti, R. and Panet, G. *Méthode MERISE, démarche et pratique*. Editions d'Organisation, 1985.

[TARJAN 72] Tarjan, R.E. Depth first search and linear algorithms. *SIAM Journal on Computing*, *1* (2). 146 - 160, 1972.

[VAN DER AALST 97a] Van-der-Aalst, W.M.P. Verification of Workflow Nets. *Application and Theory of Petri Nets, Lecture Notes in Computer Science Springer - Verlag 1248* (In P. Azema and G. Balbo, editors,). 407 - 426, 1997.

[VAN DER AALST 97b] Van-der-Aalst, W.M.P., W.M.P., D.H. and Verbeek:, H.M.W., A Petri-net-based tool to analyze Workflows. in *the Conference on Petri Nets in System Engineering* (Universität Hamburg, Germany, 1997), B. Farwer, D. Moldt and M.-O. Stehr Fachbereich Informatik, 78 - 90.

[VAN DER AALST 98a] Van-der-Aalst, W.M.P. The Application of Petri Nets to Workflow Management. *The journal of Circuits, Systems and Computers*, *7* (1). 1 - 45, 1998.

[VAN DER AALST 98b] Van-der-Aalst, W.M.P. Chapter 10: Three Good reasons for Using a Petri-net-based Workflow Management System. in al, T.W.e. ed. *Information and Process Integration in Enterprises: Rethinking documents*, Kluwer Academic, 1998.

[VAN DER AALST 98c] Van-der-Aalst, W.M.P., Michelis, G.D. and Ellis, C.A. Proceedings of Workflow Management: Net-based Concepts, Models, Techniques and Tools. C.A. ed., Eindhoven University of Technology, Eindhoven, the Netherlands, 1998.

[VAN DER AALST 98d] Van-der-Aalst, W.M.P., Basten, T., Verbeek, H.M.W., Verkoulen, P.A.C. and Voorhoeve, M., Adaptive Workflow On the interplay between flexibility and support. in *International Conference on Enterprise Information Systems*, (1998).

[VAN DER AALST 04] Van-der-Aalst, W.M.P. and Hee, K.V. *Workflow Management: Models, Methods, and Systems*, 2004.

[VERBEEK 01] Verbeek, H.M.V., Basten, T. and Aalst., W.M.P.v.d. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, *44* (4). 246 - 279, 2001.

[VINOSKI 97] Vinoski, S. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments *IEEE Communications Magazine*, 1997, 46 - 55.

[VOORHOEVE 97] Voorhoeve, M. and Van-der-Aalst, W.M.P., Ad-hoc Workflow: Problems and Solutions. in *the 8 th DEXA Workshop Conference on Database and Expert Systems Applications*, (Toulouse, France, 1997), 36 - 41.

[WAINER 01] Wainer, G., Giambiasi, N. Application of the Cell-DEVS paradigm for cell spaces modelling and simulation. *Simulation* , Vol. 71, No. 1. January 2001. pp. 22-39.

[WAINER 02] Wainer, G. CD++: a toolkit to develop DEVS models. *Software - Practice & Experience 32* (13). 1261 - 1306, 2002.

[WAINER 04] Wainer, G. DEVS tools, Carleton, 2004, http://www.sce.carleton.ca/faculty/wainer/standard/tools.htm.

[WAINER 96] Wainer, J., Weske, M., Vossen, G. and Medeiros, C., Scientific Workflow systems. in *NSF Workshop on Workflow and Process Automation*, (USA, 1996).

[WARSHALL 62] Warshall, S. A Theorem on Boolean Matrices. *Journal of the ACM*, *9* (1). 11 - 12, 1962.

[WEATHERLY 98] Weatherly, R., DoD High Level Architecture for Modeling and Simulation in *Software Engineering and Economics Conference*, (1998).

[WFMC03f 98] WfMC-TC-1011. Glossaire. D.Hollingsworth ed., 1998.

[WFMC11 99] WfMC-TC-1011. Terminology & Glossary. Hollingsworth, E.D. ed., 1999.

[WFMC25 05] WFMC-TC-1025. Workflow Process Definition Interface -- XML Process Definition Language (XPDL), 2005.

[YASPER 05] Yasper. Yasper, 2005, http://www.yasper.org/.

[YAWL 05] Yawl. Yawl, Queensland University of Technology GPO Box 2434 Brisbane Qld 4001 AUSTRALIA, 2005, http://www.yawl.fit.qut.edu.au/.

[ZACHAREWICZ 02] Zacharewicz, G., Frydman, C. and Zanni, C., Workflow/HLA Distributed Simulation Environment. in *Harbour, Maritime & Multimodal Logistics Modelling and Simulation*, (Bergeggi, Italy, 2002).

[ZACHAREWICZ 05a] Zacharewicz, G., Giambiasi, N. and Frydman, C., Improving the DEVS/HLA Environment. in *DEVS Integrative M&S Symposium, Part of the 2005 SCS Spring Simulation Multiconference, SpringSim'05*, (San Diego, CA, USA, 2005).

[ZACHAREWICZ 05b] Zacharewicz, G., Giambiasi, N. and Frydman, C., A New Algorithm for the HLA Lookahead Computing in the DEVS/HLA Environment. in *European Simulation Interoperability Workshop (EUROSIW)*, (Toulouse, France, 2005, 2005).

[ZACHAREWICZ 05c] Zacharewicz G., N.G., C. Frydman GDEVS/HLA Environment: A Time Management Improvement. in *the 17th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation*, (Paris, France, 2005).

[ZACHAREWICZ 05d] Zacharewicz, G., Giambiasi, N. and Frydman, C., Improving the Lookahead Computation in G-DEVS/HLA Environment. in *The 9-th IEEE International Symposium on Distributed Simulation and Real Time Applications (IEEE DS-RT 05)*, (Montreal, Canada, 2005), 273-282.

[ZACHAREWICZ 06a] Zacharewicz, G., Giambiasi, N. and Frydman, C. Lookahead Computation in G-DEVS/HLA Environment. *Simulation News Europe Journal (SNE)* (special issue 1 "Parallel and Distributed Simulation Methods and Environments"), 2006

[ZACHAREWICZ 06b] Zacharewicz, G., Hamri, A., Trojet, W., Frydman, C. and Giambiasi, N., G-DEVS / HLA Environment for Distributed Simulations of Workflows. in *Invited talk in: International Conference on Modeling and Simulation - Methodology, Tools, Software Applications (M&S-MTSA'06)*, (Calgary, Alberta, Canada, 2006).

[ZAJAC 03] Zajac, K., Bubak, M., Malawski, M. and Sloot, P., Towards a Grid Management System for HLA-Based Interactive Simulations. in *The 7th IEEE International Symposium on Distributed Simulation and Real-Time Applications DS-RT*, (2003), 4 - 12.

[ZEIGLER 76] Zeigler, B.P. *Theory of Modelling and Simulation*. Wiley & Sons, New York, NY, 1976.

[ZEIGLER 84] Zeigler, B.P. *Multifacetted modelling and discrete event simulation*. Academic Press Professional, Inc., San Diego, CA, 1984.

[ZEIGLER 95] Zeigler, B.P. *Object Oriented Simulation with Hierarchical, Modular Models: Selected Chapters Updated for DEVS-C++*. Originally published by Academic Press, 1990., 1995.

[ZEIGLER 98a] Zeigler, B.P. and Lee, J.S. Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment. *Enabling Technology for Simulation Science II SPIE*, *3369*. 49 - 58, 1998.

[ZEIGLER 98b] Zeigler, B.P. and Ball, G. The DEVS/HLA Distributed Simulation Environment And Its Support for Predictive Filtering, ECE Dept., UA, Tucson, AZ, 1998.

[ZEIGLER 99] Zeigler, B.P., Ball, G., Cho, H.J. and Lee, J.S., Implementation of the DEVS formalism over the HLA/RTI: Problems and solutions. in *Simulation Interoperation Workshop (SIW)*, (Orlando, FL, 1999).

[ZEIGLER 00] Zeigler, B.P., Praehofer, H. and Kim, T.G. *Theory of Modeling and Simulation*, New York, NY, 2000.

UN ENVIRONNEMENT G-DEVS/HLA : APPLICATION A LA MODELISATION ET SIMULATION DISTRIBUEE DE WORKFLOW

Résumé :
Les travaux de cette thèse portent sur :

- la proposition d'algorithmes de simulation distribuée conservative de modèles DEVS / G-DEVS,
- la définition et la réalisation d'un environnement de modélisation & simulation (M&S) G-DEVS compatible HLA implémentant les algorithmes proposés,
- l'application de l'environnement à la M&S de Workflow.

Dans un premier temps, nous avons introduit un composant coordinateur racine G-DEVS distribué, incluant un algorithme de communication avec le RTI HLA basé sur le mécanisme de synchronisation conservative et utilisant un Lookahead positif. Nous avons ensuite proposé deux algorithmes originaux pour le calcul d'un Lookahead relatif à l'état courant d'un modèle G-DEVS. Ces algorithmes, basés sur l'analyse du domaine de variation de la fonction « durée de vie » du modèle, augmentent les performances de la simulation distribuée comme l'illustrent les expériences menées.

Basé sur ces approches, nous avons développé un environnement de M&S distribué G-DEVS / HLA. Cet environnement a été intégré à une application de Workflow. Les possibilités offertes par l'environnement ont été illustrées par l'étude de cas réels d'entreprises.

Mots clés : DEVS, GDEVS, Simulation Distribuée, Synchronisation Conservative, Lookahead, HLA, Gestion du temps HLA, FEDEP HLA, Workflow

A G-DEVS / HLA ENVIRONMENT: APPLICATION TO DISTRIBUTED MODELLING AND SIMULATION OF WORKFLOW

Abstract:
The researches of this thesis are divided in:

- the proposition of conservative distributed simulation algorithms of models DEVS / G-DEVS,
- the definition and the realization of a G-DEVS modeling and simulation (M&S) environment HLA compliant implementing the proposed algorithms,
- the application of the environment in the M&S of Workflow.

First, we introduced a G-DEVS distributed root coordinator component, including an algorithm of communication with the RTI HLA based on the mechanism of conservative synchronization and using positive Lookahead. Then, we proposed two original algorithms for the computation of Lookahead concerning the current state of a G-DEVS model. These algorithms, based on the analysis of the variation domain of the "lifetime" function of the model, increase the performances of the distributed simulation as the led experiments illustrate it.

Based on these approaches, we developed a distributed G-DEVS / HLA M&S environment. This environment was integrated into an application of Workflow. The possibilities offered by the environment were illustrated by the study of company's real cases of Workflow.

Key words: DEVS, GDEVS, Distributed Simulation, Conservative Synchronization, Lookahead, HLA, HLA time Management, HLA FEDEP, Workflow